Unification in Sort Theories and its Applications

Christoph Weidenbach

MPI–I–94–211 March 1994

Author's Address

Max-Planck-Institut für Informatik, Im Stadtwald,
W-6600 Saarbrücken 11, Germany.
Email: weidenb@mpi-sb.mpg.de

Publication Notes

Submitted to the Journal "Annals of Mathematics and Artificial Intelligence".

Acknowledgements

This work was supported by the ESPRIT project 6471 MEDLAR of the European Community.

I would like to thank my colleages at the Max-Planck-Institut for many helpful comments on this paper.

Abstract

In this article I investigate the properties of unification in sort theories. The usual notion of a sort consisting of a sort symbol is extended to a set of sort symbols. In this language sorted unification in elementary sort theories is of unification type finitary. The rules of standard unification with the addition of four sorted rules form the new sorted unification algorithm. The algorithm is proved sound and complete. The rule based form of the algorithm is not suitable for an implementation because there is no control and the used data structures are weak. Therefore we transform the algorithm into a deterministic sorted unification procedure. For the procedure sorted unification in pseudo-linear sort theories is proved decidable.

The notions of a sort and a sort theory are developed in a way such that a standard calculus can be turned into a sorted calculus by replacing standard unification with sorted unification. To this end sorts may denote the empty set. Sort theories may contain clauses with more than one declaration and may change dynamically during the deduction process. The applicability of the approach is exemplified for the resolution and the tableau calculus.

Keywords

Sorts, Sorted Unification, Sorted Resolution, Sorted Tableau

1 Introduction

The advantages of sorts in automated reasoning are well known [15, 21, 18, 4, 27, 10, 14, 22, 31, 1, 5, 12, 29]. The difference between standard first-order calculi and their sorted counterparts amounts to an extended language and special reasoning mechanisms for sorts, monadic predicates. The language is extended by attaching sorts to variables, i.e. the domain of variables can be restricted to subsets (the interpretation of sorts) of the domain (the interpretation of the top sort). This additional information is exploited in the sorted unification algorithm which replaces standard unification. Sorted unification is performed with respect to a sort theory (a set of formulae containing sorts).

In the standard setting [27, 10, 22, 1] a sort theory $\mathcal{L}$ consists of a set of unit clauses each containing a declaration (atoms with monadic predicate as top symbol). An example for such a sort theory is

$$\mathcal{L} \quad = \quad \{S(a), S(f(x_S)), T(a), T(f(y_T))\}$$

Now to solve the unification problem [23]

$$\Gamma \quad = \quad \{x_S = y_T\}$$

the unification problem must be standard solved and variables have to be *weakened* (instantiated by appropriate terms) until the components of the corresponding unifier are *well-sorted*. A component $x_S \mapsto t$ is well-sorted if in every model of $\mathcal{L}$ we have $S(t)$. As $\mathcal{L}$ has a special structure there exists always a free model for $\mathcal{L}$, the free term algebra $\mathcal{T}$ of well-sorted terms. Computations in this algebra are efficient. Following the above example, well-sorted terms of sort S and T are

$$\begin{array}{lcl} \mathcal{T}_S & = & \{x_1, x_2, \ldots, a, f(a), f(f(a)), \ldots\} \\ \mathcal{T}_T & = & \{y_1, y_2, \ldots, a, f(a), f(f(a)), \ldots\} \end{array}$$

where all x_j have sort S (this is written $\mathcal{S}(x_j) = S$), $\mathcal{S}(y_j) = T$, and there are infinitely many variables in each set. As neither $x_S \in \mathcal{T}_T$ nor $y_T \in \mathcal{T}_S$ the unification problem Γ is not sorted solved. At least one of the variables must be weakened. The sort theory allows weakening using the declarations $S(a)$, $T(a)$, and $S(f(x_S))$, $T(f(y_T))$, respectively. Using the first two declarations we get the well-sorted unifier

$$\sigma_1 \quad = \quad \{x_S \mapsto a, y_T \mapsto a\}$$

The second two declarations lead to the still unsolved unification problem

$$\Gamma' \quad = \quad \{x_S = f(x'_S), y_T = f(y'_T), x'_S = y'_T\}$$

Again the four declarations selected before must be applied to $x'_S = y'_T$ resulting in infinitely many well-sorted mgu's

$$\begin{aligned}\sigma_2 &= \{x_S \mapsto f(a), y_T \mapsto f(a)\}\\ \sigma_3 &= \{x_S \mapsto f(f(a)), y_T \mapsto f(f(a))\}\\ &\vdots\end{aligned}$$

The first extension I make to this standard setting allows the domain of variables to be restricted to a set of sorts, interpreted as the intersection of the sorts. For the example the free algebra then is

$$\begin{aligned}\mathcal{T}_S &= \{x_1, x_2, \ldots, z_1, z_2, \ldots, a, f(a), f(f(a)), \ldots\}\\ \mathcal{T}_T &= \{y_1, y_2, \ldots, z_1, z_2, \ldots, a, f(a), f(f(a)), \ldots\}\\ \mathcal{T}_{\{S,T\}} &= \{z_1, z_2, \ldots, a, f(a), f(f(a)), \ldots\}\end{aligned}$$

where $\mathcal{S}(x_j) = S$, $\mathcal{S}(y_j) = T$, and $\mathcal{S}(z_j) = \{S, T\}$. Instead of the set $\{S\}$ we simply write the sort symbol S. Having the extended sort language the solution to Γ is one single most general well-sorted unifier

$$\sigma = \{x_S \mapsto z_{\{S,T\}}, y_T \mapsto z_{\{S,T\}}\}$$

The new notion allows a finite set of unifiers in cases where previous approaches [27, 22, 1, 5] lead to infinite sets.

The second extension investigated in this article needs further motivation. The idea of sorted unification is to provide a sorted unification algorithm and a notion of a sort theory such that a standard calculus using unification can be turned into a sorted calculus by replacing standard unification with sorted unification. In addition, I don't want to impose any restrictions on the sort theory, i.e. any monadic predicate can be used as a sort. In my previous work [28, 30] the problem is solved for the resolution calculus. It turned out not sufficient to consider unit clauses consisting of one single declaration as sort theories. All clauses consisting of declarations form the sort theory. From every clause in the sort theory exactly one declaration is selected for the construction of well-sorted terms, hence for sorted unification. The other declarations are attached to the selected declaration as a condition. Using this notion of a sort theory standard resolution can be extended to sorted resolution by replacing standard unification with sorted unification. The new notion of a sort theory makes it necessary to extend the notion of well-sortedness to a notion of conditional well-sortedness. For example having the unsatisfiable clause database Δ

Δ:	
(1)	$S(a) \vee R(a)$
(2)	$S(f(x_1))$
(3)	$Q(y_1, y_1)$
(4)	$\neg Q(a, y_2)$
(5)	$P(x_2, f(a))$
(6)	$\neg P(a, x_3)$

where $\mathcal{S}(x_i) = S, \mathcal{S}(y_i) = R$. If the first clause $S(a) \vee R(a)$ is not included in the sort theory no sorted resolution step is possible. Thus one of the following sort theories must be chosen

$$\begin{aligned} \mathcal{L}_1 &= \{(S(a), \{R(a)\}), S(f(x_1))\} \\ \mathcal{L}_2 &= \{(S(a), \{R(a)\}), S(f(x_1))\} \end{aligned}$$

In $\mathcal{L}_1$ the declaration $S(a)$ is selected for the generation of well-sorted terms and $R(a)$ is attached as a condition. The sets of conditional well-sorted terms for $\mathcal{L}_1$ are:

$$\begin{aligned} \mathcal{T}_S &= \{x_i, z_i, (a, \{R(a)\}), (f(a), \{R(a)\}), (f(f(a)), \{R(a)\}), \ldots\} \\ \mathcal{T}_R &= \{y_i, z_i\} \\ \mathcal{T}_{\{S,R\}} &= \{z_i\} \end{aligned}$$

where $\mathcal{S}(z_i) = \{S, R\}$. Now solving the unification problem

$$\Gamma = \{x_3 = f(a), x_2 = a\}$$

the result is the conditional well-sorted mgu

$$\sigma = (\{x_3 \mapsto f(a), x_2 \mapsto a\}, \{R(a)\})$$

Thus choosing $\mathcal{L}_1$ sorted resolution between the clauses (5) and (6) is possible with resolvent $R(a)$. If we choose $\mathcal{L}_2$ sorted resolution between the clauses (3) and (4) is possible with resolvent $S(a)$. Sorted resolution is described in full detail in Section 6.

An outline of the article is this. We start with a small section on foundations (Section 2), where the syntax, semantics of the logic and some technical notions needed later on are explained. The section on standard unification (Section 3) introduces the general notions for standard unification and a rule based version of the Robinson [20] unification algorithm. These notions are then extended in the section on sorted unification (Section 4) to their sorted counterparts. The notion of conditional expressions and conditional well-sortedness are introduced. The sorted unification algorithm, an algorithm for deciding conditional well-sortedness and an algorithm for the computation of empty sorts are given. Upper bounds for the time complexity of these algorithms are established.

As a result of the extended sort language it is shown that unification in elementary sort theories is decidable and of unification type finitary. In Section 5 the sorted unification algorithm given by a set of non-deterministic rules in Section 4 is transformed into a deterministic sorted unification procedure. To this end the notion of a unification problem is extended. This makes it possible to prove sorted unification decidable for pseudo-linear sort theories, the most general result known to date for a decidable class. The section on applications (Section 6) demonstrates that the previously developed notions and algorithms make sense. Sorted unification is applied to the standard resolution calculus and the standard tableau calculus. The article is finished with a discussion on the achieved results and related work, Section 7.

2 Foundations

Preliminaries: A reflexive and transitive relation $\leq$ on a set A is called a *quasi-ordering*. A quasi-ordering $\leq$ naturally generates an equivalence relation $\equiv$, such that $a \equiv b$ iff $a \leq b$ and $b \leq a$. The *equivalence class* in A with respect to $\equiv$ is denoted as $[a]_{\equiv}$. We use $a < b$ to denote that $a \leq b$ but not $a \equiv b$.

An element a is minimal in A iff for all $b \in A$: $b \leq a$ implies $a \leq b$. A quasi-ordering is *linear*, iff $a \leq b$ or $b \leq a$ for all elements. It is *well-founded*, iff every chain has a minimal element. A quasi-ordering satisfying $a \leq b$ and $b \leq a$ implies $a = b$ (antisymetrie) is called a *partial ordering*.

A *multiset* over a set A is a function M from A to the natural numbers. Intuitively, $M(a)$ specifies the number of occurrences of a in M. We say that a is an element of M if $M(a) > 0$. The union, intersection, and difference of multisets are defined by the identities $M_1 \cup M_2 = M_1(x) + M_2(x)$, $M_1 \cap M_2 = \min(M_1(x), M_2(x))$, and $M_1 \setminus M_2 = \max(0, M_1(x) - M_2(x))$. We often use a set-like notation to describe multisets. If we have a well-founded partial ordering on the elements of a multiset M, then we can construct recursively a well-founded multiset-ordering on multisets as follows: $M > N$ if for some $a \in M$ and $b_i \in N$ $i = 1, \ldots, n$: $a > b_i$ and $M \setminus \{a\} > N \setminus \{b_1, \ldots, b_n\}$ (e.g. see Dershowitz [8]).

Syntax: The standard first-order signature $\Sigma = (V_\Sigma, F_\Sigma, P_\Sigma)$ where V_Σ, F_Σ are infinite sets of variable and function symbols, respectively and P_Σ is a finite set of predicate symbols is generalized in the following way. Monadic predicates are also called *base sorts*. S_Σ is the set of all base sorts ($S_\Sigma \subseteq P_\Sigma$). A *sorts* is an element of 2^{S_Σ}. For $\emptyset \in 2^{S_\Sigma}$ we write $\top$ the usual sort attached to standard variables, $\top \notin \Sigma$ and mean the top sort. As S_Σ is finite, 2^{S_Σ} is finite too. The function $\mathcal{S}$, $\mathcal{S}\colon V_\Sigma \to 2^{S_\Sigma}$ maps variables to sorts such that for each sort $S \in 2^{S_\Sigma}$ there are infinitely many variables x with $\mathcal{S}(x) = S$. In examples it is indicated that a variable x has sort S by writing x_S. If S is the singleton set $S = \{T\}$ we often write x_T instead of $x_{\{T\}}$. If not stated otherwise variables not annotated with a sort have sort $\top$, the top sort.

Terms, literals, clauses, formulae, and substitutions are defined in the usual (standard) way. With $\forall(\mathcal{F})$ the universal closure of a formula $\mathcal{F}$ is denoted. Literals built from base sorts are called *sort literals*. Positive sort literals are called *declarations*. A clause only consisting of declarations is called a *declaration clause*. A declaration $S(t)$ is called a *subsort* declaration if t is a variable. Otherwise it is called a *term* declaration. If t has the form $f(x_1, \ldots, x_n)$ then $S(t)$ is called a *function* declaration.

With $DOM(\sigma) := \{x \mid x\sigma \neq x\}$ we denote the finite domain of a substitution σ and $COD(\sigma) := \{x\sigma \mid x \in DOM(\sigma)\}$. Specific substitutions are described by their variable-term pairs, e.g. $\{x \mapsto a\}$ denotes the substitution x maps to a.

The function $\mathcal{V}$ maps terms, formulae and sets of such expressions to their variables. In order to select subterms of a given term t we use occurrences [16]. An *occurrence* is a word over $\mathbb{N}$. Let Λ denote the empty word. Then we define the set of occurrences $Occ(t)$ of a term t as follows: (i) the empty word Λ is in $Occ(t)$ (ii)

$i.\pi$ is in $Occ(t)$ iff $t = f(t_1, \ldots, t_n)$ and $\pi \in Occ(t_i)$. The *depth* of a term t denoted by $Depth(t)$ is defined as the maximal length of an occurrence in $Occ(t)$. The *size* of a term is the number of symbols in it, or equivalently the number of occurrences in $Occ(t)$, $Size(t) := |Occ(t)|$.

A term t is called *pseudo-linear* if every occurrence of a variable x in t has the same length. A term t is called *semi-linear* [26] if for every two occurrences $i_1...i_k$, $j_1...j_k$ of a variable x in t the top symbols of the terms $t\backslash i_1...i_l$ and $t\backslash j_1...j_l$ are the same for $1 \leq l \leq k$.

Semantics: As the only extension of standard syntax are sorted variables it is sufficient to present the semantics for these variables. The semantics of sorted variables is given by the two relativization rules

$$\begin{array}{lcl} \forall x\, \mathcal{F} & \rightarrow & \forall y\, ((S_1(y) \wedge \ldots \wedge S_n(y)) \Rightarrow \mathcal{F}\{x \mapsto y\}) \\ \exists x\, \mathcal{F} & \rightarrow & \exists y\, (S_1(y) \wedge \ldots \wedge S_n(y) \wedge \mathcal{F}\{x \mapsto y\}) \end{array}$$

where $\mathcal{S}(x) \neq \top$, $\mathcal{S}(x) = \{S_1, \ldots, S_n\}$, $\mathcal{S}(y) = \top$ and y does not occur in the formula $\mathcal{F}$. Variables of sort $\top$ (the top sort) have the same semantics than standard variables.

An alternative semantics would be to extend standard interpretations $\Im$ to sorted interpretations $\Im_{\mathcal{S}}$ by modifying the rules for the quantifiers. A sorted interpretation $\Im_{\mathcal{S}}$ is like a standard interpretation $\Im$ except that:

$$\begin{array}{lll} \Im_{\mathcal{S}} \models \forall x\, \mathcal{F} & \text{iff} & \text{for all } a \in \bigcap\limits_{T \in \mathcal{S}(x)} \Im_{\mathcal{S}}(T) \text{ we have } \Im_{\mathcal{S}}[x/a] \models \mathcal{F} \\ \Im_{\mathcal{S}} \models \exists x\, \mathcal{F} & \text{iff} & \text{there is an } a \in \bigcap\limits_{T \in \mathcal{S}(x)} \Im_{\mathcal{S}}(T) \text{ with } \Im_{\mathcal{S}}[x/a] \models \mathcal{F} \end{array}$$

where $\Im_{\mathcal{S}}[x/a]$ is like $\Im_{\mathcal{S}}$ except that it maps x to a. It should be clear that the sorted semantics and the semantics given by the relativization rules are equivalent with respect to sentences.

3 Standard Unification

For the standard case a lot of efficient unification procedures are known, e.g. [19, 17, 2]. For reasons of simplicity we will present a rule based version of the Robinson [20] unification procedure following [17]. We use the standard notions for unification [23].

Definition 3.1 (Notions for Unification) A substitution σ is called a *matcher* from s to t if $s\sigma = t$, where s and t are two expressions (terms, atoms, literals). t and s are called *unifiable*, iff there exists a substitution σ such that $t\sigma = s\sigma$. In this case the substitution σ is called a *unifier* of t and s. A unifier σ of two expressions t and s is called an *mgu (most general unifier)*, iff for every unifier λ of t and s there exists a substitution τ, such that $\sigma\tau = \lambda$. If t and s are two expressions, then $\Gamma = \{t = s\}$ is called the *unification problem* for t and s. A substitution σ *solves*

Tautology	$\dfrac{x = x \cup \Gamma}{\Gamma}$
Decomposition	$\dfrac{\{f(t_1, \ldots, t_n) = f(s_1, \ldots, s_n)\} \cup \Gamma}{\{t_1 = s_1, \ldots, t_n = s_n\} \cup \Gamma}$
Application	$\dfrac{\{x = t\} \cup \Gamma}{\{x = t\} \cup \Gamma\{x \mapsto t\}}$ if x is a variable, $x \notin \mathcal{V}(t)$, and $x \in \mathcal{V}(\Gamma)$
Orientation	$\dfrac{\{t = x\} \cup \Gamma}{\{x = t\} \cup \Gamma}$ if x is a variable and t a non-variable term
Clash	$\dfrac{\{f(t_1, \ldots, t_n) = g(s_1, \ldots, s_m)\} \cup \Gamma}{\text{STOP.FAIL}}$ if $f \neq g$
Cycle	$\dfrac{\{x = t\} \cup \Gamma}{\text{STOP.FAIL}}$ if $x \in \mathcal{V}(t)$

Table 1: The Rules of Standard Unification

a unification problem $\Gamma = \{t_1 = s_1, \ldots, t_n = s_n\}$, iff $t_1\sigma = s_1\sigma, \ldots, t_n\sigma = s_n\sigma$. A unification problem Γ is called *solved*, iff $\Gamma = \{x_1 = t_1, \ldots, x_n = t_n\}$ where the x_i are variables, $x_i \notin \mathcal{V}(t_j)$ and $x_i \neq x_j$ for every i and j.

Algorithm 3.2 (A Standard Unification Algorithm) The input of the algorithm is a unification problem Γ, which is changed by the following six rules until it is solved or the problem is found to be unsolvable:

If $\Gamma = \{t_1 = s_1, \ldots, t_n = s_n\}$ is a unification problem, then the above unification algorithm always terminates on Γ. If the algorithm stops with failure there is no substitution σ solving Γ. Otherwise Γ is solved and the corresponding substitution σ is an idempotent mgu of the term pairs $(t_1, s_1), \ldots, (t_n, s_n)$.

Lemma 3.3 Let t, s be two unifiable, pseudo-linear terms. If s and t share variables, we assume that occurrences of common variables also have the same length. If σ is the mgu of s and t, then $Depth(t\sigma) = \max(Depth(s), Depth(t))$.

Proof: By induction on the number n of different variables occurring in s or t.

The base cases $n = 0$, $n = 1$ and $n = 2$ can be easily proved.

Now assume $n + 1$ different variables. We apply exhaustively the rules Tautology, Decomposition, and Orientation to the unification problem $\Gamma = \{s = t\}$ resulting in $\Gamma' = \{x_1 = t_1, \ldots, x_n = t_n\}$. Let $\lambda = \{x_1 \mapsto t_1\}$, $t' = t\lambda$, and $s' = s\lambda$. Then $\max(Depth(s), Depth(t)) = \max(Depth(s'), Depth(t'))$ because s and t are pseudo-linear. In addition s' and t' are also pseudo-linear and contain n different variables. By induction hypothesis for the mgu τ of s' and t' we have $Depth(t'\tau) = \max(Depth(s'), Depth(t'))$. Thus from $\lambda\tau = \sigma$ and the above we have $Depth(t\sigma) = \max(Depth(s), Depth(t))$. □

4 Unification in Sort Theories

4.1 Conditional Well-Sorted Expressions

Definition 4.1 (Conditional Expressions) A pair $(S(t), C)$ is called a *conditional declaration* (*conditional term*, *conditional substitution*) if C is a finite set of literals and $S(t)$ a declaration (term, substitution). A conditional expression (t, C) is called *ground* if t is ground.

We always assume that the set $\mathcal{L}$ of conditional declarations is finite and all conditional declarations are variable disjoint. The set $\mathcal{L}$ is called a *sort theory*. All following notions and definitions refer to a fixed sort theory $\mathcal{L}$.

Definition 4.2 (Conditional Well-Sorted Terms) The set of conditional well-sorted terms (abbreviated by cws. terms) $\mathcal{T}_S$ of sort S is recursively defined by:

(i) For every variable $x \in V_\Sigma$ and every sort $S \subseteq \mathcal{S}(x)$, $(x, \emptyset) \in \mathcal{T}_S$.

(ii) For every conditional declaration $(S(t), C) \in \mathcal{L}$, $(t, C) \in \mathcal{T}_S$. Note that there are only declarations for base sorts.

(iii) For every cws. term $(t, C) \in \mathcal{T}_S$ $(S \neq \top)$, substitution $\sigma := \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, with $(t_i, C_i) \in \mathcal{T}_{\mathcal{S}(x_i)}$ for all i, $(\bigcup_i C_i) \subseteq D$ for some finite set of literals D, $(t\sigma, C\sigma \cup D) \in \mathcal{T}_S$.

(iv) For cws. terms $(t, C_i) \in \mathcal{T}_{S_i}$ $(1 \leq i \leq n)$, finite set of literals D, $(\bigcup_i C_i) \subseteq D$, $(t, D) \in \mathcal{T}_{\{S_1, \ldots, S_n\}}$.

(v) For every term t and finite set of literals D we have $(t, D) \in \mathcal{T}_\top$.

We define $\mathcal{T}_S^\emptyset := \{(t, \emptyset) \mid (t, \emptyset) \in \mathcal{T}_S\}$. $\mathcal{T}_{S,\mathrm{gr}}$ is the restriction of $\mathcal{T}_S$ to ground terms. A sort S is called *empty* if there is no cws. ground term $(t, C) \in \mathcal{T}_S$, or equivalently if $\mathcal{T}_{S,\mathrm{gr}} = \emptyset$. We always have $\mathcal{T}_S^\emptyset \subseteq \mathcal{T}_S$ and $\mathcal{T}_{S,\mathrm{gr}} \subset \mathcal{T}_S$.

For the unification algorithm it is useful to define a binary relation $\sqsubseteq$ which denotes the subsort relationship. If S and T are sorts, then we define $S \sqsubseteq T$ iff

there exists a variable x with $\mathcal{S}(x) = S$ and $x \in \mathcal{T}_T$. Note that if there exists one variable $x \in \mathcal{T}_T$ there are infinitely many variables of sort S in $\mathcal{T}_T$. The relation $S \sqsubseteq T$ implies $\mathcal{T}_S \subseteq \mathcal{T}_T$. In addition, we have $t \in \mathcal{T}_{\{S,T\}}$ iff $t \in \mathcal{T}_S$ and $t \in \mathcal{T}_T$ for two arbitrary sorts S, T.

Sometimes we are not interested in the condition part of a conditional expression. We say that $t \in \mathcal{T}_S$ if there is a set of literals C such that $(t, C) \in \mathcal{T}_S$. Similarly we say that a declaration $S(t) \in \mathcal{L}$ if there is a set of literals C such that $(S(t), C) \in \mathcal{L}$. If not necessary or if the conditional part is the empty set we do not mention the conditional part of a conditional expression.

Example 4.3 (Conditional Well-Sorted Terms) Consider the sort theory

$$\mathcal{L} \quad = \quad \{(S(a), \{P(a)\}), (S(y), \{Q(y)\}), T(a), T(f(x))\}$$

If a conditional declaration (term) has an empty conditional part only the term is written. It is assumed throughout this example that $\mathcal{S}(x_i) = S$ and $\mathcal{S}(y_i) = T$. Now Definition 4.2 is applied. The case (i) gives

$$\begin{array}{rcl} \mathcal{T}_S & \supset & \{x_1, x_2, \ldots, z_1, z_2, \ldots\} \\ \mathcal{T}_T & \supset & \{y_1, y_2, \ldots, z_1, z_2, \ldots\} \\ \mathcal{T}_{\{S,T\}} & \supset & \{z_1, z_2, \ldots\} \end{array}$$

with $\mathcal{S}(z_i) = \{S, T\}$. Case (ii) of Definition 4.2 extends the sets by terms occurring in the declarations:

$$\begin{array}{rcl} \mathcal{T}_S & \supset & \{(a, \{P(a)\}), (y, \{Q(y)\})\} \\ \mathcal{T}_T & \supset & \{a, f(x)\} \end{array}$$

Now case (iii) can be applied. We have $(y, \{Q(y)\}) \in \mathcal{T}_S$. For both substitutions $\sigma = \{y \mapsto a\}$ and $\tau = \{y \mapsto f(x)\}$ the components are well-sorted, i.e. $a \in \mathcal{T}_T$ and $f(x) \in \mathcal{T}_T$. $\mathcal{T}_S$ is extended by the new terms $(y\sigma, \{Q(y)\sigma\})$ and $(y\tau, \{Q(y)\tau\})$:

$$\mathcal{T}_S \quad \supset \quad \{(a, \{Q(a)\}), (f(x), \{Q(f(x))\})\}$$

Now all terms in $\mathcal{T}_S$ can be substituted for x in $f(x) \in \mathcal{T}_T$:

$$\mathcal{T}_T \quad \supset \quad \{(f(a), \{P(a)\}), (f(a), \{Q(a)\}), (f(y), \{Q(y)\})\}$$

This process can be continued infinitely many times. Applying case (iv) using the occurrences of $f(a)$ in $\mathcal{T}_S$ and $\mathcal{T}_T$ we get:

$$\mathcal{T}_{\{S,T\}} \quad \supset \quad \{(f(a), \{P(a), Q(a), Q(f(a))\}), (f(a), \{Q(a), Q(f(a))\})\}$$

Case (v) means that an arbitrary conditional term is included in $\mathcal{T}_\top$. The sorts S, T and $\{S, T\}$ are not empty and

$$\begin{array}{rcl} \mathcal{T}_S^\emptyset & = & \{x_i, z_i\} \\ \mathcal{T}_T^\emptyset & = & \{y_i, z_i, a, f(x_i)\} \end{array}$$

Depending on the declarations occurring in $\mathcal{L}$ the following theories are distinguished. $\mathcal{L}$ is called *elementary* if every term declaration is a function declaration. $\mathcal{L}$ is called *semi-linear* if every term occurring in a term declaration is semi-linear and there are no cycles with respect to $\sqsubseteq$. $\mathcal{L}$ is called *pseudo-linear* if every term occurring in a term declaration is pseudo-linear. $\mathcal{L}$ is called *regular* if all subsort declarations have an empty condition part, $\sqsubseteq$ is a partial ordering and every term t has a unique least sort with respect to $\sqsubseteq$. Note that this notion is called preregular by Goguen and Meseguer [14].

Lemma 4.4 (Soundness of Cws. Terms) Let $\mathcal{L} = \{(S_1(t_1), \{K_{1,1}, \ldots, K_{1,n_1}\}), \ldots, (S_m(t_m), \{K_{m,1}, \ldots, K_{m,n_m}\})\}$ be a sort theory. For every interpretation $\Im_{\mathcal{S}}$ with $\Im_{\mathcal{S}} \models \forall(S_1(t_1) \vee K_{1,1} \vee \ldots \vee K_{1,n_1}) \wedge \ldots \wedge \forall(S_m(t_m) \vee K_{m,1} \vee \ldots \vee K_{m,n_m})$ and every cws. term $(t, \{L_1, \ldots, L_k\}) \in \mathcal{T}_S$, $S \neq \top$, $S = \{S_1, \ldots, S_n\}$, we have $\Im_{\mathcal{S}} \models \forall((S_1(t) \wedge \ldots \wedge S_n(t)) \vee L_1 \ldots \vee L_k)$.

Proof: By structural induction according to Definition 4.2. □

Lemma 4.5 (Completeness of Cws. Terms) Let $\mathcal{L}$ be a sort theory where all declarations have an empty condition part, whence $\mathcal{T}_S = \mathcal{T}_S^{\emptyset}$ for every sort S. Then $\Im_{\mathcal{S}}^{\mathrm{gr}}$ given by $\Im_{\mathcal{S}}^{\mathrm{gr}}(S) = \mathcal{T}_{S,\mathrm{gr}}$ for all base sorts S is the initial model in the family of all $\mathcal{L}$ models and $\Im_{\mathcal{S}}^{\mathrm{fr}}$ given by $\Im_{\mathcal{S}}^{\mathrm{fr}}(S) = \mathcal{T}_S$ is the free model in the family of all $\mathcal{L}$ models.

Proof: Straightforward extension of the proof given by Schmidt-Schauß [22]. □

Definition 4.6 (Conditional Well-Sorted Substitutions) A conditional substitution $\sigma^{\mathbf{c}} = (\sigma, C)$ is called *conditional well-sorted* if for every $x_i \in DOM(\sigma)$, there is a cws. term $(x_i\sigma, C_i) \in \mathcal{T}_{\mathcal{S}(x_i)}$ and $(\bigcup_i C_i) \subseteq C$.

A cws. renaming $\sigma^{\mathbf{c}} = (\sigma, \emptyset)$ is a cws. substitution such that $COD(\sigma)$ consists of variables only, σ is injective on $DOM(\sigma)$ and $\mathcal{S}(x) = \mathcal{S}(x\sigma)$ for all $x \in DOM(\sigma)$.

The composition of two well-sorted substitutions can be computed by $\tau^{\mathbf{c}}\sigma^{\mathbf{c}} := (\tau\sigma, K\sigma \cup C)$, where $\sigma^{\mathbf{c}} = (\sigma, C)$, $\tau^{\mathbf{c}} = (\tau, K)$. The result of the composition is again a cws. substitution. Thus the set of all cws. substitutions builds a monoid. The set of all conditional well-sorted substitutions is denoted by SUB. A cws. substitution $\sigma^{\mathbf{c}} \in SUB$ is called *empty* iff there exists no $\lambda^{\mathbf{c}} \in SUB$ such that $\sigma^{\mathbf{c}}\lambda^{\mathbf{c}}$ is a ground substitution.

Example 4.7 (Conditional Well-Sorted Substitutions) Again we consider the sort theory of Example 4.3, $\mathcal{S}(x_i) = S$, $\mathcal{S}(y_i) = T$:

$$\mathcal{L} \quad = \quad \{(S(a), \{P(a)\}), (S(y), \{Q(y)\}), T(a), T(f(x))\}$$

The two substitutions σ and τ are cws. substitutions:

$$\begin{array}{lcl} \sigma^{\mathbf{c}} & = & \{y_1 \mapsto a\} \\ \tau^{\mathbf{c}} & = & (\{y \mapsto f(y_1)\}, \{Q(y_1)\}) \end{array}$$

Note that these substitutions would be also cws. if the conditional part is extended by additional literals. The composition $\tau\sigma$ is

$$\tau^{\mathbf{c}}\sigma^{\mathbf{c}} \quad = \quad (\{y \mapsto f(a), y_1 \mapsto a\}, \{Q(a)\})$$

Definition 4.8 Let $s^{\mathbf{c}} = (s, C), t^{\mathbf{c}} = (t, K)$ be two cws. terms. Then

(i) $s^{\mathbf{c}} \geq_{\mathcal{L}} t^{\mathbf{c}}$ iff there exists $(\sigma, D) \in SUB$ such that $s = t\sigma$ and $(K\sigma \cup D) \subseteq C$. In this case we call (σ, D) an *instantiating substitution* of $t^{\mathbf{c}}$ to $s^{\mathbf{c}}$ and we call $s^{\mathbf{c}}$ an $\mathcal{L}$-*instance* of $t^{\mathbf{c}}$.

(ii) $s^{\mathbf{c}} \equiv_{\mathcal{L}} t^{\mathbf{c}}$ iff $s^{\mathbf{c}} \geq_{\mathcal{L}} t^{\mathbf{c}}$ and $t^{\mathbf{c}} \geq_{\mathcal{L}} s^{\mathbf{c}}$.

As $\geq_{\mathcal{L}}$ is a quasi-ordering on cws. terms the relation $\equiv_{\mathcal{L}}$ is an equivalence relation.

Lemma 4.9 For every sort $S \in 2^{S_\Sigma}$, $S \neq \top$, $S = \{S_1, \ldots, S_n\}$, and every non-variable term $s^{\mathbf{c}} \in \mathcal{T}_S$ there exist term declarations $(S_i'(t_i), C_i) \in \mathcal{L}$ with $S_i' \sqsubseteq S_i$ and a cws. substitution $(\sigma, D) \in SUB$ such that $(t_i\sigma, (\bigcup C_i)\sigma \cup D \cup E\sigma) \leq_{\mathcal{L}} s^{\mathbf{c}}$ for all i, where E is the set of conditions coming from the subsort declarations which establish $S_i' \sqsubseteq S_i$.

Proof: By structural induction using Definition 4.2 and the fact that SUB is a monoid. □

Lemma 4.10 Let t be a term and let $S \neq \top$ be a sort.

(i) There exists a finite set of cws. terms $Cond(t, S) = \{(t, C_1), \ldots, (t, C_n)\}$ with $(t, C_i) \in \mathcal{T}_S$ for all i and for each cws. term $(t, C) \in \mathcal{T}_S$ there is a term $(t, C_i) \in Cond(t, S)$ with $(t, C) \geq_{\mathcal{L}} (t, C_i)$.

(ii) There exists a finite set of cws. ground terms $Cond(S) = \{(t_1, C_1), \ldots, (t_n, C_n)\}$ with $(t_i, C_i) \in \mathcal{T}_S$ for all i and for each cws. ground term $(t, C) \in \mathcal{T}_S$ there is a ground term $(t_i, C_i) \in Cond(S)$ and a cws. renaming $\sigma^{\mathbf{c}} = (\sigma, \emptyset)$ with $C_i\sigma \subseteq C$.

Proof:

(i) By induction on the structure of terms. If t is a constant and $(t, C) \in \mathcal{T}_S$, for an arbitrary set of literals C, then by Lemma 4.9 there are term declarations in $\mathcal{L}$ which can be used to build a more general term than (t, C). As $\mathcal{L}$ is finite there are only finitely many different terms. If t is a variable, then $\mathcal{S}(t) \sqsubseteq S$. As there are only finitely many subsort declarations in $\mathcal{L}$ the set $Cond(t, S)$ is finite. If t is a compound term, Lemma 4.9 and fact that $\mathcal{L}$ is finite can be used again and the induction hypothesis is applied to the components of σ (see Lemma 4.9).

(ii) The set exists because $\mathcal{L}$ is finite, the sets of ground terms are recursively enumerable and the subset ordering modulo renaming is well founded on finite sets of literals. □

There exists no effective algorithm for the computation of common ground terms for two base sorts. The question whether two base sorts share a common ground term is undecidable in general [22]. The set $Cond(S)$ is effectively computable for base sorts if the sorts of all variables occurring in $\{x \mid x \in \mathcal{V}(t), (S(t), C) \in \mathcal{L}\}$ are base sorts. In this case the time complexity of computing $Cond(S)$ for all base sorts is at most $O(m^3 * k)$ where $m = |\mathcal{L}|$ and $k = max(\{|C| \mid (S(t), C) \in \mathcal{L}\})$. The idea is to start with all ground declarations and collect the conditions and ground terms for the respective sorts. These sorts are non-empty with respect to the conditions. Then this result is propagated recursively to all declarations where all sorts attached to the variables in the declaration are already known non-empty. The corresponding conditions are collected, ground terms built, and using the subset relationship redundant sets of conditions can be removed. The cardinality of the maximal non-redundant set of conditions is bound by $m * k$ and for the search and propagation we need at most m^2 steps.

As every term has top sort $\top$, $Cond(t, \top) = \{t\}$ and $Cond(\top) = \{a\}$ for an arbitrary constant a. In addition, sort computation is trivial for sort $\top$.

Example 4.11 Again we consider the sort theory of Example 4.3,

$$\mathcal{L} \quad = \quad \{(S(a), \{P(a)\}), (S(y), \{Q(y)\}), T(a), T(f(x))\}$$

Then examples for $Cond(t, S)$ are

$$\begin{array}{lcl} Cond(f(a), T) & = & \{(f(a), \{Q(a)\}), (f(a), \{P(a)\})\} \\ Cond(f(x), T) & = & \{f(x)\} \\ Cond(f(x), S) & = & \{(f(x), \{Q(f(x))\})\} \end{array}$$

and the sets $Cond(S)$ and $Cond(T)$ are

$$\begin{array}{lcl} Cond(T) & = & \{a\} \\ Cond(S) & = & \{(a, \{P(a)\}), (a, \{Q(a)\})\} \end{array}$$

Algorithm 4.12 (Sort Computation) Let t be a term and $S \neq \top$ a sort. Then $t \in \mathcal{T}_S$ with $S = \{S_1, \ldots, S_n\}$ iff there exist sorts $T_i \in Sorts(t)$ with $T_i \sqsubseteq S_i$ for all i. The function *Sorts* can be computed as follows:

(i) if t is a variable, then $Sorts(t) = \{\mathcal{S}(t)\}$

(ii) if t is a constant, then $Sorts(t) = \{S \mid S(t) \in \mathcal{L}\}$

(iii) if t is a compound term $f(t_1, \ldots, t_n)$, then $Sorts(t) = \{S \mid S(f(s_1, \ldots, s_n)) \in \mathcal{L}$, there exists a standard matcher σ with $f(s_1, \ldots, s_n)\sigma = f(t_1, \ldots, t_n)$ and for each component $x_i \mapsto t'_i$ of σ, $\mathcal{S}(x_i) = \{S_1, \ldots, S_n\}$ there are sorts $T_i \in Sorts(t'_i)$ with $T_i \sqsubseteq S_i\}$

The above algorithm can also be used to answer queries of the form $t \in \mathcal{T}_S^{\emptyset}$ by only considering declarations with empty condition part.

The time complexity of the algorithm for a query $t \in \mathcal{T}_S$ is at most $O(n^2 * m)$ where $n = \mathit{Size}(t)$ and $m = |L|$. The idea is to start with the computation of the sorts of subterms of depth 0 and then successively continue for subterms with depth greater 0. Schmidt-Schauß [22] proved that sort computation is quasi-linear. His result doesn't subsume our result because Schmidt-Schauß considered constant time for all operations concerning the sort theory. This was reasonable because his sort theory is a static part of the signature. In the approach presented here the theory typically changes during deduction, if sorted unification is applied to a calculus (see Section 6).

Example 4.13 (Sort Computation) The query $f(f(x_1)) \in \mathcal{T}_S$ is answered with respect to the sort theory of Example 4.3 ($\mathcal{S}(x_i) = S$, $\mathcal{S}(y_i) = T$):

$$\mathcal{L} \quad = \quad \{(S(a), \{P(a)\}), (S(y), \{Q(y)\}), T(a), T(f(x))\}$$

$\mathit{Depth}(x_1) = 0$ thus the sort computation is started with x_1. As we have $\mathcal{S}(x_1) = S$ the result for x_1 is

$$\mathit{Sorts}(x_1) \quad = \quad \{S\}$$

Following case (i) of Definition 4.12 the term $f(x_1)$ is the only term of depth 1. The only applicable declaration (case (iii)) is $T(f(x))$ with standard matcher $\sigma = \{x \mapsto x_1\}$. Of course we have $S \in \mathit{Sorts}(x)$, whence

$$\mathit{Sorts}(f(x_1)) \quad = \quad \{T\}$$

For $f(f(x_1))$ again declaration $T(f(x))$ with standard matcher $\sigma = \{x \mapsto f(x_1)\}$ can be used. We have $T \in \mathit{Sorts}(f(x_1))$ and $T \sqsubseteq S$, because $(S(y), \{Q(y)\}) \in \mathcal{L}$.

$$\mathit{Sorts}(f(f(x_1))) \quad = \quad \{T\}$$

Now we know that $f(f(x_1)) \in \mathcal{T}_S$ because $T \sqsubseteq S$.

Definition 4.14 Let $W \subseteq V_\Sigma$ and $\sigma^{\mathbf{c}} = (\sigma, C), \tau^{\mathbf{c}} = (\tau, K)$, $\sigma^{\mathbf{c}}, \tau^{\mathbf{c}} \in \mathit{SUB}$

(i) $\sigma^{\mathbf{c}} \geq_{\mathcal{L}} \tau^{\mathbf{c}}[W]$ iff there exists a $\lambda^{\mathbf{c}} = (\lambda, D) \in \mathit{SUB}$ with $x\sigma = x\tau\lambda$ for all $x \in W$ and $(K\lambda \cup D) \subseteq C$
In this case we call $\lambda^{\mathbf{c}}$ an *instantiating substitution* of $\tau^{\mathbf{c}}$ to $\sigma^{\mathbf{c}}$ and we call $\sigma^{\mathbf{c}}$ an $\mathcal{L}$*-instance* of $\tau^{\mathbf{c}}$ modulo W.

(ii) $\sigma^{\mathbf{c}} \equiv_{\mathcal{L}} \tau^{\mathbf{c}}$ iff $\sigma^{\mathbf{c}} \geq_{\mathcal{L}} \tau^{\mathbf{c}}$ and $\tau^{\mathbf{c}} \geq_{\mathcal{L}} \sigma^{\mathbf{c}}$.

4.2 Unification Theory

We shortly recall the basic notions of unification theory [23] for the case of sort theories. Let Γ be a unification problem. A cws. substitution $\sigma^{\mathbf{c}} = (\sigma, C)$ solves Γ iff for every equation $(s = t) \in \Gamma$ we have $s\sigma = t\sigma$. $\sigma^{\mathbf{c}}$ is called a cws. *unifier* of Γ. The set of all unifiers of Γ is written $U_{\mathcal{L}}(\Gamma)$, which is a left ideal in the substitution monoid SUB, since $U_{\mathcal{L}}(\Gamma) = SUB \circ U_{\mathcal{L}}(\Gamma)$.

We define the *complete set of unifiers* called $cU_{\mathcal{L}}(\Gamma)$ of a unification problem Γ as a set of cws. unifiers satisfying

(i) $cU_{\mathcal{L}}(\Gamma) \subseteq U_{\mathcal{L}}(\Gamma)$ (correctness)

(ii) $\forall \delta^{\mathbf{c}} \in U_{\mathcal{L}}(\Gamma)\ \exists \sigma^{\mathbf{c}} \in cU_{\mathcal{L}}(\Gamma) : \delta^{\mathbf{c}} \geq \sigma^{\mathbf{c}}[\mathcal{V}(\Gamma)]$ (completeness)

The base set $\mu U_{\mathcal{L}}(\Gamma)$, called the *set of most general unifiers*, is defined as a complete set of unifiers satisfying in addition

(iii) $\forall \sigma^{\mathbf{c}}, \tau^{\mathbf{c}} \in cU_{\mathcal{L}}(\Gamma) :$ if $\sigma^{\mathbf{c}} \geq \tau^{\mathbf{c}}[\mathcal{V}(\Gamma)]$ then $\sigma^{\mathbf{c}} = \tau^{\mathbf{c}}$ (minimality)

Based on the cardinality of μU, we can classify sort theories according to the following *unification hierarchy*. A sort theory $\mathcal{L}$ is of type:

unitary	if $\mu U_{\mathcal{L}}(\Gamma)$ exists and has at most one element for all unification problems Γ
finitary	if $\mu U_{\mathcal{L}}(\Gamma)$ exists and is finite for all unification problems Γ
infinitary	if $\mu U_{\mathcal{L}}(\Gamma)$ exists and is infinite for some unification problem Γ
nullary	if $\mu U_{\mathcal{L}}(\Gamma)$ does not exist for some unification problem Γ

4.3 The Sorted Unification Algorithm

Definition 4.15 (Notions for Sorted Unification) In order to define sorted unification, we have to extend Definition 3.1. A unification problem $\Gamma = \{x_1 = t_1, \ldots, x_n = t_n\}$ is called *sorted solved* if Γ is solved and for every variable x_i we have $t_i \in \mathcal{T}_{\mathcal{S}(x_i)}$.

Lemma 4.16 The ordering $\geq_{\mathcal{L}}\ [W]$ is well founded on SUB for a finite set of variables W.

Proof: We have to show that there are no infinite chains of cws. substitutions $(\sigma_1, C_1) >_{\mathcal{L}} (\sigma_2, C_2) >_{\mathcal{L}} \ldots$ Choose a weight function $w{:}\,\Sigma \to \mathbb{N}$ such that $w(S) > w(T)$ if $S \sqsubset T$ or $S \sqsubset^{\emptyset} T$ for sorts $S, T \in 2^{S_\Sigma}$, $w(x){:=}\, w(\mathcal{S}(x))$ for all variables $x \in V_\Sigma$, $w(f){:=}\, max\{w(S) \mid S \in 2^{S_\Sigma}\} + 1$ for all $f \in F_\Sigma$. The relation $\sqsubset^{\emptyset}$ is the restriction of $\sqsubset$ to subsort declarations with an empty condition part. Note that $S \sqsubset^{\emptyset} T$ implies $S \sqsubset T$ or $S \equiv T$. As $\mathcal{L}$ and 2^{S_Σ} are finite the relation $\sqsubset$

is well founded, therefore the function w always exists. w may assign arbitrary natural numbers to predicate symbols. w can be extended to terms, literals, and sets of such expressions by $w(f(t_1, \ldots, t_n)) := w(f) + \sum_i w(t_i)$ and $w(\{E_1, \ldots, E_n\}) := \sum_i w(E_i)$. The weight of a cws. substitution is defined as $w((\sigma, C)) := w(DOM(\sigma)) + w(COD(\sigma)) + w(C)$.

Now we show $(\sigma, C) >_{\mathcal{L}} (\tau, D)$ implies $w((\sigma, C)) > w((\tau, D))$. Wlog. we assume $DOM(\sigma) \subseteq W$, $DOM(\tau) \subseteq W$ and thus $DOM(\sigma) = DOM(\tau)$. If $(\sigma, C) >_{\mathcal{L}} (\tau, D)$ either $|C| > |D|$ or some subterms in C or $COD(\sigma)$ are replaced by variables in D or $COD(\tau)$, respectively, or some variables in C or $COD(\sigma)$ are replaced by more general variables in D or $COD(\tau)$, respectively. All these cases imply $w((\sigma, C)) > w((\tau, D))$. As the sets $COD(\sigma)$, $COD(\tau)$, C, D, W are all finite, we conclude $\geq_{\mathcal{L}} [W]$ is well founded. □

As a consequence of Lemma 4.16 we have that for every finite set Γ of equations, there exists a minimal, complete set of unifiers $\mu U_{\mathcal{L}}(\Gamma)$.

Algorithm 4.17 (The Sorted Unification Algorithm) The input of the algorithm is a unification problem Γ, which is changed by four sorted rules (see Table 2) and the six standard rules of Algorithm 3.2 (see Table 1) until it is sorted solved or no rule is applicable or the problem is found to be unsolvable.

In order to compute a cws. substitution from a sorted solved unification problem, we have to do the following. Let $\Gamma = \{x_1 = t_1, \ldots, x_n = t_n\}$ be the sorted solved unification problem, then $\sigma := \{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$ is the corresponding unifier. $\sigma^{\mathbf{c}} := (\sigma, C)$ is a cws. mgu if we have $(t_i, C_i) \in \mathcal{T}_{\mathcal{S}(x_i)}$ for all i and $C = \bigcup_i C_i$. Thus from a sorted solved unification problem we may compute several (but only finitely many, see Lemma 4.10) cws. mgu's.

As the sorted unification algorithm is an extension of the standard unification algorithm with the sorted rules, every sorted unifier is a standard instance of the standard unifier. This was pointed out by Schmidt-Schauß [22] already. Frisch and Cohn used this insight for a more abstract presentation of sorted unification [12].

Note that the unification algorithm unifies terms with respect to all cws. substitutions, including empty substitutions. Thus the problem of empty sorts is separated from the unification problem.

Lemma 4.18 (Soundness of the Unification Algorithm) If the unification algorithm computes a cws. mgu $\sigma^{\mathbf{c}}$ for a unification problem Γ, then $\sigma^{\mathbf{c}}$ solves Γ.

Proof: Can be easily proved for the rules. The proof is done by showing that every cws. substitution solving the problem after the application of a rule solves the original problem. □

Lemma 4.19 (Completeness of the Unification Algorithm) If $\lambda^{\mathbf{c}}$ is a cws. substitution which solves Γ, then the unification algorithm computes a cws. mgu $\sigma^{\mathbf{c}}$ which solves Γ and $\lambda^{\mathbf{c}} \geq_{\mathcal{L}} \sigma^{\mathbf{c}}[\mathcal{V}(\Gamma)]$.

Sorted Fail	$\dfrac{\{x = f(t_1, \ldots, t_n)\} \cup \Gamma}{\text{STOP.FAIL}}$ if $x \notin \mathcal{V}(f(t_1, \ldots, t_n))$, $f(t_1, \ldots, t_n) \notin \mathcal{T}_{\mathcal{S}(x)}$, $\mathcal{S}(x) = \{S_1, \ldots, S_n\}$ and there are no conditional declarations $S_i'(f(s_{i,1}, \ldots, s_{i,n})) \in \mathcal{L}$, $S_i' \sqsubseteq S_i$
Subsort	$\dfrac{\{x = y\} \cup \Gamma}{\{y = x\} \cup \Gamma}$ if $x \in \mathcal{T}_{\mathcal{S}(y)}$ and $y \notin \mathcal{T}^{\emptyset}_{\mathcal{S}(x)}$
Common Subsort	$\dfrac{\{x = y\} \cup \Gamma}{\{x = z\} \cup \{y = z\} \cup \Gamma}$ if $x \notin \mathcal{T}^{\emptyset}_{\mathcal{S}(y)}$, $y \notin \mathcal{T}^{\emptyset}_{\mathcal{S}(x)}$, and $\mathcal{S}(z) = \mathcal{S}(x) \cup \mathcal{S}(y)$
Weakening	$\dfrac{\{x = f(t_1, \ldots, t_n)\} \cup \Gamma}{\{x = f(s_{1,1}, \ldots, s_{1,n})\} \cup \{t_1 = s_{i,1}, \ldots, t_n = s_{i,n}\} \cup \Gamma}$ if $x \notin \mathcal{V}(f(t_1, \ldots, t_n))$, $f(t_1, \ldots, t_n) \notin \mathcal{T}^{\emptyset}_{\mathcal{S}(x)}$, $\mathcal{S}(x) = \{S_1, \ldots, S_n\}$ and for each S_i there is a conditional declaration $S_i'(f(s_{i,1}, \ldots, s_{i,n})) \in \mathcal{L}$, $S_i' \sqsubseteq S_i$

Table 2: The Sorted Rules of Sorted Unification

Proof: Let $\sigma^{\mathbf{c}} = (\sigma, C)$ be an idempotent most general unifier, i.e. $\sigma^{\mathbf{c}} \in \mu U_{\mathcal{L}}(\Gamma)$ with $DOM(\sigma) = \mathcal{V}(\Gamma)$. For the proof we will split Γ into two disjoint parts: Γ_U and Γ_{WO}. Γ_U contains the unsolved equations and is initialized with Γ. Γ_{WO} contains the worked off equations, i.e. equations already processed by the rules of the unification algorithm. Γ_{WO} is initialized with the empty set.

As well-founded complexity measure $\mu(\sigma, \Gamma)$ we use the multiset of all term depths in $\Gamma_U\sigma$. The idea of the proof is to show that there exists a pair (σ', Γ'), such that Γ can be transformed into Γ' by one step of the unification algorithm and σ' is a mgu of Γ' that is equal to σ on old variables and extends σ to new variables, furthermore $\mu(\sigma', \Gamma') < \mu(\sigma, \Gamma)$.

If $\mu(\sigma', \Gamma')$ is minimal, i.e. $\Gamma_U = \emptyset$, then the set of equations is sorted solved and we are ready. It should be clear that in this case Γ_{WO} is sorted solved and by using Lemma 4.10 we can compute the desired mgu by restricting the domain of σ' to $\mathcal{V}(\Gamma)$.

Now we show that there is always a step of the unification algorithm that reduces the measure $\mu(\sigma, \Gamma)$. First we argue that the rule Application (see Algorithm 3.2) does not increase the measure. The rule does not change the depths of terms is $\Gamma\sigma$, since from $x\sigma = t\sigma$ we obtain $\sigma\{x \mapsto t\} = \sigma$, since σ is idempotent.

We go through the cases for equations $s = t$ in Γ_{WO}:

(i) Case $s = t$, where neither s nor t is a variable. Then by step Decomposition we reduce $\mu(\sigma, \Gamma)$ without changing the set of solutions.

(ii) Case $x = f(t_1, \ldots, t_n)$. Then $x \notin \mathcal{V}(f(t_1, \ldots, t_n)$ because $x\sigma = f(t_1, \ldots, t_n)\sigma$. If $x\sigma = f(t_1, \ldots, t_n)$, then we have $f(t_1, \ldots, t_n) \in \mathcal{T}_{\mathcal{S}(x)}$, move the equation to Γ_{WO} and are done. If $x\sigma \neq f(t_1, \ldots, t_n)$ then $f(t_1, \ldots, t_n) \notin \mathcal{T}^{\emptyset}_{\mathcal{S}(x)}$ as σ is an idempotent mgu of Γ. Let $\mathcal{S}(x) = \{S_1, \ldots, S_n\}$. By Lemma 4.9 there exist declarations $S_i'(f(s_{i,1}, \ldots, s_{i,n}))$ with $S_i' \sqsubseteq S_i$ and substitution $\tau^{\mathbf{c}} = (\tau, E)$ with $f(s_{i,1}, \ldots, s_{i,n})\tau = f(t_1, \ldots, t_n)\sigma$. We use the rules Weakening, Decomposition, Orientation and Application to obtain a new equation system Γ'. Since the equation $x = f(s_{1,1}, \ldots, s_{1,n})$ is sorted solved, we have $\mu(\sigma \cup \tau, \Gamma') < \mu(\sigma, \Gamma)$, since the depth of $f(t_1, \ldots, t_n)\sigma$ is larger than all term depths of $t_j\sigma$ and $s_{i,j}\tau$. Furthermore $\sigma \cup \tau$ is a solution of Γ' with $\sigma \cup \tau = \sigma[\mathcal{V}(\Gamma)]$.

(iii) Case $x = y$. If $x\sigma = y$ or $y\sigma = x$ we can shift the equation to Γ_{WO} after applying rule Subsort if necessary. If this is not possible we apply rule Common Subsort and move the sorted solved equations $x = z$ and $y = z$ to Γ_{WO}. With $\tau = \{z \mapsto x\sigma\}$ we have $\mu(\sigma \cup \tau, \Gamma') < \mu(\sigma, \Gamma)$ and $\sigma \cup \tau = \sigma[\mathcal{V}(\Gamma)]$

□

Example 4.20 (Infinitely Many mgu's) Consider the following sort theory $\mathcal{L}$ and unification problem Γ, where $\mathcal{S}(x_i) = S$:

$$\begin{array}{lcl} \mathcal{L} & = & \{S(g(g(x))), S(g(a)), S(a)\} \\ \Gamma & = & \{x_1 = g(x_2)\} \end{array}$$

We have $\mathcal{T}_S = \{g(g(x)), g(a), a, g^i(a), g^{2i}(x), x_i\}$ with $i \geq 2$. Now we apply the rules of sorted unification. Γ is standard solved but not sorted solved because $g(x_2) \notin \mathcal{T}_S$. The only applicable rule is Weakening using the declarations $S(g(g(x_3)))$ and $S(g(a))$ resulting in the two unification problems

$$\begin{array}{lcl} \Gamma_1 & = & \{x_1 = g(g(x_3)), x_2 = g(x_3)\} \\ \Gamma_2 & = & \{x_1 = g(a), x_2 = a\} \end{array}$$

respectively. Γ_2 is sorted solved because $\{g(a), a\} \subseteq \mathcal{T}_S$. Γ_1 is not sorted solved because $g(x_3) \notin \mathcal{T}_S$. Again only rule Weakening using the declarations $S(g(g(x_4)))$ and $S(g(a))$ is applicable. The two new unification problems after the application of standard unification (see Algorithm 3.2) are

$$\begin{array}{lcl} \Gamma_3 & = & \{x_1 = g(g(g(x_4))), x_2 = g(g(x_4)), x_3 = g(x_4)\} \\ \Gamma_4 & = & \{x_1 = g(g(a)), x_2 = g(a), x_3 = a\} \end{array}$$

respectively. Γ_4 is sorted solved because $\{g(g(a)), g(a), a\} \subseteq \mathcal{T}_S$. Γ_3 is not sorted solved because $g(x_4) \notin \mathcal{T}_S$. The algorithm stucks in a cycle. Rule Weakening can be further applied always leading to a solved problem using $S(g(a))$ and an unsolved problem containing an equation of the form $x_i = g(x_{i+1})$ using $S(g(g(x)))$ renamed. The example demonstrates that unification in sort theories may lead to infinitely many mgu's.

Lemma 4.21 (Properties of $\mathcal{L}$-Unification)

(i) $\mathcal{L}$-Unification is of unification type infinitary.

(ii) If $\mathcal{L}$ is elementary then $\mathcal{L}$-Unification is decidable and of unification type finitary. In addition $\mathcal{L}$-Unification is NP-complete and the number of unifiers may grow exponentially with the size of terms to be unified.

Proof: (i) The set $\mu U_{\mathcal{L}}(\Gamma)$ exists for every finite set Γ of equations and finite set $\mathcal{L}$ of conditional declarations (Lemma 4.16). Example 4.20 shows a sort theory and a unification problem such that $\mu U_{\mathcal{L}}(\Gamma)$ is infinite. Note that the above sort theory $\mathcal{L}$ is not elementary (because $S(g(g(x))) \in \mathcal{L}$) but linear (therefore pseudo-linear).

(ii) The only crucial rule of sorted unification concerning termination and decidability is the rule Weakening. Assume that the rule is applied to an equation $x = t$ occurring in the standard solved unification problem Γ. After the application of Weakening again standard unification is performed. Let t_i be the terms resulting from the application of Weakening and standard unification to $x = t$. As $\mathcal{L}$ is elementary and because of Lemma 3.3 we have $\mathit{Depth}(t) > \mathit{Depth}(t_i)$ or a variable occurring non-pseudo-linear in t is removed. As t contains only finitely many variables the number of terms generated by the Weakening rule is finite. Thus $\mathcal{L}$-unification is decidable and of unification type finitary for elementary sort theories.

NP-completeness can be shown in the same way than for the sorted unification algorithm of Schmidt-Schauß [22]. The idea is to reduce the satisfiability problem of propositional logic to $\mathcal{L}$-unification. This gives also examples where the number of unifiers may grow exponentially in the size of terms to be unified. □

The result on elementary theories is not new. Schmidt-Schauß [22] showed unification in elementary sort theories to be decidable but infinitary due to his weaker sort language. Uribe [26] gave an algorithm which computes at most finitely many solved forms for a given unification problem in a semi-linear sort theory. However, there is an important difference between our and Uribe's approach. Our unification algorithm works in all cases whereas Uribe's algorithm only works for semi-linear theories. Thus the result means we have an algorithm which works in any case and has the desired properties for the sub case of elementary sort theories.

Now a semi-decision algorithm for the computation of empty sorts is presented. The algorithm consists of two phases: a fast preprocessing phase there sorts are marked which can be easily detected to be non-empty and a processing phase there it is checked whether a specific sort is empty or not.

Algorithm 4.22 (Computation of Empty Sorts) The input of the algorithm *Empty* is a sort $S = \{S_1, \ldots, S_n\}$. The algorithm returns *True* if S is empty and *False* otherwise.
Preprocessing:

(i) For each ground declaration $S(t) \in \mathcal{L}$, mark S to be non-empty.

(ii) For each declaration $S'(t') \in \mathcal{L}$ where all sorts attached to variables in t' are marked non-empty, mark S' to be non-empty. Step (ii) is repeated until all declarations have been checked and no new sort is detected to be non-empty.

Processing:

(i) If S is marked to be non-empty then $Empty(S) = False$. If S is marked to be empty then $Empty(S) = True$.

(ii) If $S = \{S_1, \ldots, S_n\}$ and if there are conditional declarations $S_i'(f(t_{i,1}, \ldots, t_{i,n})) \in \mathcal{L}$, $S_i' \sqsubseteq S_i$ such that the $f(t_{i,1}, \ldots, t_{i,n})$ are sorted unifiable with unifier σ and for all sorts $T \in \{\mathcal{S}(x) \mid x \in \mathcal{V}(COD(\sigma))\}$ we have $Empty(T) = False$ then $Empty(S) := False$ and S is marked non-empty else $Empty(S) := true$ and S is marked empty.

The algorithm is correct and complete which can be easily seen using Lemma 4.19, Lemma 4.18 and Lemma 4.10. If the marking is done by attaching the appropriate sets of conditions, the algorithm can also be used to compute the set of conditions which guarantees a sort to be non-empty. These conditions are needed if sorted unification is applied to a calculus (see Section 6). The time complexity of the

preprocessing phase is at most $O(n^2)$ where $n = |\mathcal{L}|$. This phase has only to be computed once for a sort theory $\mathcal{L}$. It decides the non-emptiness for all base sorts which only depend on declarations where only base sorts are attached to the variables occurring in the declarations.

Note that the second phase is recursive and is only a semi-decision algorithm if non-base sorts occur. There are several possibilities to show that the empty sort problem is undecidable. One possibility is to reduce it to Schmidt-Schauß's empty sort problem which is known to be undecidable. Instead of checking whether $\{S_1, \ldots, S_n\}$ is empty we apply Schmidt-Schauß sorted unification to the unification problem $\Gamma = \{x_1 = x_2,\ x_2 = x_3,\ \ldots,\ x_{n-1} = x_n\}$ with $\mathcal{S}(x_i) = S_i$. It is sufficient for the undecidability result to consider sort theories where only base sorts are attached to variables.

In addition Comon [6] pointed out that sort theories correspond to finite bottom-up tree automaton. Using this relationship it can be shown that the empty sort problem is undecidable for pseudo-linear signatures [7, 3]. In order to show this result it is sufficient to consider non-linearities on the second level of a term. For semi-linear signatures Uribe [26] showed that the empty sort problem is decidable.

Example 4.23 (Computation of Empty Sorts) We apply Algorithm 4.22 to the the non-base sort $\{S, T\}$ with respect to the sort theory

$$\mathcal{L} \quad = \quad \{(S(b), \{P(a)\}), (S(f(f(y_2))), \{Q(y_2)\}), T(a), T(f(y_1))\}$$

where $\mathcal{S}(y_i) = T$. We compute $\mathit{Empty}(\{S, T\})$. The preprocessing phase detects S and T to be non-empty using the declarations $T(a)$ and $S(b)$. Note that this does not imply $\{S, T\}$ to be non-empty because the two terms a and b are different. Following case (ii) the only applicable declarations are $S(f(f(y_2)))$ and $T(f(y_1))$. The terms $f(f(y_2))$ and $f(y_1)$ have to be unified. The only cws. unifier is $\sigma = \{y_1 \mapsto f(y_2)\}$. As we have $\mathit{Empty}(T) = \mathit{False}$, following case (i) we conclude $\mathit{Empty}(\{S, T\}) = \mathit{False}$.

4.4 Examples for Sorted Unification

In the following we present more examples illustrating the different phenomena coming up with the sorted unification algorithm. First, we give an example with respect to a sort theory where all declarations have an empty condition part. Second, we target on the question of what happens if the condition part of declarations is not empty.

We start with the following elementary sort theory and unification problem Γ ($\mathcal{S}(x_i) = S$, $\mathcal{S}(y_i) = T$):

$$\begin{array}{lcl} \mathcal{L} & = & \{S(a), T(a), S(f(x)), T(f(y))\} \\ \Gamma & = & \{x_1 = y_1\} \end{array}$$

We have

$$\begin{array}{lcl} \mathcal{T}_S & \supset & \{a, f^i(a), f^i(x_j), x_i, z_i\} \\ \mathcal{T}_T & \supset & \{a, f^i(a), f^i(y_j), y_j, z_i\} \end{array}$$

where $i, j \geq 1$ and $\mathcal{S}(z_i) = \{S, T\}$. Applying the rule Common Subsort to Γ results in the following unification problem:

$$\Gamma_1 \quad = \quad \{x_1 = z, y_1 = z\}$$

Problem Γ_1 is sorted solved. Especially $\mathcal{T}_{\{S,T\}}$ contains infinitely many ground terms $f^i(a)$. The example shows that we get one unifier where sorted approaches only considering base sorts compute infinitely many unifiers [22, 10].

The above sort theory doesn't contain any conditional declarations, therefore $\mathcal{T}_S = \mathcal{T}_S^{\emptyset}$ for every sort S. The next sort theory contains a conditional declaration:

$$\begin{array}{lcl} \mathcal{L} & = & \{S(a), S(g(a)), (S(g(x)), \{P(x)\}), S(f(g(x)))\} \\ \Gamma & = & \{x_1 = f(x_2)\} \end{array}$$

Examples for cws. terms of sort S are:

$$\begin{array}{lcl} \mathcal{T}_S & \supset & \{a, g(a), (g(x), \{P(x)\}), \quad f(g(x)), \quad f(g^i(x_j)), \quad f(g^i(a)), \\ & & (f(g(g(x))), \{P(x)\}), (f(g^3(x)), \{P(x), P(g(x))\}), x_j\} \\ \mathcal{T}_S^{\emptyset} & \supset & \{a, g(a), f(g(a)), f(g(g(a))), f(g(x_j)), x_j\} \end{array}$$

where $i > 1$. The only applicable rule of sorted unification is Weakening

$$\Gamma_1 \quad = \quad \{x_1 = f(g(x_3)), x_2 = g(x_3)\}$$

Γ_1 is sorted solved yielding the cws. unifier

$$\sigma_1 \quad = \quad (\{x_1 \mapsto f(g(x_3)), x_2 \mapsto g(x_3)\}, \{P(x_3)\})$$

Nevertheless Weakening is still applicable to $x_2 = g(x_3)$ using the declaration $S(g(a))$

$$\Gamma_2 \quad = \quad \{x_1 = f(g(x_3)), x_2 = g(a), x_3 = a\}$$

Γ_1 is also sorted solved yielding the cws. unifier

$$\sigma_2 \quad = \quad (\{x_1 \mapsto f(g(a)), x_2 \mapsto g(a)\}, \emptyset)$$

In fact $\mu U_{\mathcal{L}}(\Gamma) = \{\sigma_1, \sigma_2\}$ for the above sort theory $\mathcal{L}$ and unification problem Γ. The example shows that the rules of sorted unification are also applied to sorted solved unification problems until the problem is sorted solved with respect to $\mathcal{T}^{\emptyset}$.

4.5 Sorted Matching

A cws. substitution (σ, C) is called a *cws. matcher* from (t, D) to (s, E) if $t\sigma = s$ and $D\sigma \cup C = E$. Sorted matching is decidable. Every cws. matcher is also a standard matcher. Therefore standard matching can be applied and then Lemma 4.10 can be used to check whether an appropriate set of conditions for the components of the matcher exists.

Sorted matching is the needed operation to decide subsumption, the most important reduction rule for resolution based calculi (see Section 6). As a cws. matcher may introduce new literals, for subsumption sorted matching is tested with respect to $\mathcal{T}^{\emptyset}$. In this case sorted matching is polynomially decidable, because first the standard matcher has to be computed and then each component is checked for conditional well-sortedness. Note that even if subsumption is checked with respect to $\mathcal{T}^{\emptyset}$ it is more powerful than standard subsumption. Consider the following database of clauses Δ where $\mathcal{S}(x_i) = S$ and $\mathcal{S}(y_i) = T$.

Δ:	
(1)	$T(a)$
(2)	$S(y)$
(3)	$Q(x_1, y_1)$
(4)	$Q(y_2, a)$

The sort theory $\mathcal{L}$ corresponding to these clauses consists of $\mathcal{L} = \{T(a), S(y)\}$ (for more details see Section 6). With respect to $\mathcal{L}$, the clause (3) subsumes clause (4) with cws. matcher

$$\sigma \quad = \quad \{x_1 \mapsto y_2, y_1 \mapsto a\}$$

The standard formalization of Δ (see the relativization rules in Section 2) is Δ'

Δ':	
(1)	$T(a)$
(2)	$\neg T(z) \vee S(z)$
(3)	$\neg S(z_1) \vee \neg T(z_2) \vee Q(z_1, z_2)$
(4)	$\neg T(z_3) \vee Q(z_3, a)$

where $\mathcal{S}(z_i) = \top$, i.e. all variables are standard variables. In the standard formalization subsumption is not applicable.

5 Implementation of Sorted Unification

The sorted unification algorithm (Algorithm 4.17) is presented by a set of nondeterministic rules. This form is suitable to proof correctness and completeness of

the algorithm but not for an implementation and further refinements. The rules have to be carefully investigated in order to solve the non-determinism and to obtain an efficient algorithm. The following observations have to be taken into account:

(i) The rules of standard unification are a subset of the rules.

(ii) The rule Subsort may be applicable infinitely many times to an equation.

(iii) The rules Subsort and Common Subsort may be both applicable to an equation.

(iv) Application of the rule Weakening may produce several new unification problems.

(v) An equation $x = t$ may be solved, i.e. $t \in \mathcal{T}_{\mathcal{S}(x)}$ but the sorted rules are still applicable to $x = t$.

(vi) If for an equation $x = t$ we have $t \in \mathcal{T}^{\emptyset}_{\mathcal{S}(x)}$, then none of the sorted rules is applicable.

(vii) Sorted unification is of unification type infinitary.

(i) means that we can split the sorted unification task into two subtasks. Standard unification which can be efficiently performed and application of the sorted rules. It is sufficient to apply rule Subsort at most once to an equation. Thus the application of rule Subsort may lead to at most two unification problems (ii). The rules Subsort and Common Subsort have to be checked independently if applied (iii). The algorithm needs to keep track of a set of unification problems (iv) and (ii). Whether an equation (a unification problem) is sorted solved has to be checked independently from the application of the sorted rules (v). well-sortedness with respect to $\mathcal{T}^{\emptyset}$ is a sufficient condition for an equation not to be further processed (vi). The algorithm may not terminate (vii). Therefore we need some kind of resource bounding and the possibility to store and continue intermediate states of sorted unification.

The key to an efficient implementation is a good representation of a unification problem. In the theory this is just a set of equations. However, the above considerations show that this is not sufficient. The proof of Lemma 4.19 suggests an extended representation. A unification problem consists of a set Γ_{WO} of worked off equations and a set of Γ_U of unsolved equations. This representation allows for an additional mechanism, cycle checking. If an equation already worked off occurs again (modulo renaming) in Γ_U (for example after the application of Weakening), the unification algorithm loops. This means that either the actual unification problem can not be sorted solved or it produces infinitely many mgu's. Checking for cycles allows to decide sorted unification in pseudo-linear sort theories (Lemma 5.4). In addition, the algorithm on empty sorts (Algorithm 4.22) can also be extended with cycle checking.

All theses considerations are implemented by the following algorithms. A unification problem is a pair (Γ_{WO}, Γ_U). This representation allows to store and continue

intermediate states. If a cycle is detected, the unification problem is suspended. If the corresponding equation could be solved by a different rule the unification problem is resumed (Algorithm 5.1 and line 6 of Algorithm 5.2). Resuming a suspended problem amounts to check whether the cycle still occurs even if the solved equations are disregarded. This is done by marking solved equations in Γ_{WO} and then checking for cycles with respect to unmarked equations. If such a cycle occurs the unification problem remains suspended. Otherwise is is resumed. The number of applications of the sorted rules is limited by the value of the input parameter *Resource*. The task of sorted unification is split into standard unification and sorted rule application. Sorted rule application is split into the processing of exactly one equation (Algorithm 5.2) and the general administration of all problems (Algorithm 5.3). These are divided into solved problems (PS), suspended problems (PP) and unsolved problems (PU).

We assume a unification algorithm for standard unification (see Definition 3.2). There are a lot of efficient unification algorithms known, e.g. see [2, 19, 17]. The standard unification algorithm is called "Standard_Unification" and gets a unification problem Γ as input. The output is a pair (Δ, *Value*) where *Value* is either *Solved* or *Fail* and if *Value* is *Solved*, then Δ is the solved form of Γ (see Definition 3.1).

The algorithms are presented in an informal programming language. The bodies of **If** , **While** , and **For** statements are indicated by indentation. For example the body of the **While** loop in algorithm Sorted_Unification consists of the lines three to sixteen. The **Return** statement immediately exits the function. The value of the function is the argument of the **Return** statement.

Algorithm 5.1 (Resume_Suspended_Problems(PP,Γ)) The input of the algorithm (see Table 3) is a set PP of suspended unification problems and a sorted solved set of equations Γ. The output is the set of problems which can be resumed because the equation which was the reason for the suspension is solved in Γ.

Algorithm 5.2 (Sorted_Rule_Application(Γ_{WO},Γ_U)) The input of the algorithm (see Table 4) is a set Γ_{WO} of worked off equations and a set Γ_U of unsolved equations.

Algorithm 5.3 (Sorted_Unification(PU,PP,*Resource*)) The input of the algorithm (see Table 5) is a set PU of unsolved unification problems and a set PP of suspended unification problems. The third parameter *Resource* is an integer which limits the maximal number of applications of Sorted_Rule_Application before Sorted_Unification terminates.

It can be easily seen that algorithm Sorted_Unification is sound and complete. In its processing of unification problems it basically follows the ideas given in the proof of Lemma 4.19.

Resume_Suspended_Problems(PP,Γ)

1. $Result := \emptyset$
2. **For** each pair (Γ_{WO}, Γ_U) in PP **Do**
3. Mark all equations in Γ_{WO} which are renamed versions of equations in Γ.
4. **If** Γ_U does not contain an equation which occurs unmarked in Γ_{WO} modulo renaming **Then** $Result := Result \cup \{(\Gamma_{WO}, \Gamma_U)\}$
5. **Return**($Result$)

Table 3: Algorithm Resume_Suspended_Problems

Lemma 5.4 (Properties of $\mathcal{L}$-Unification)
If $\mathcal{L}$ is pseudo-linear then $\mathcal{L}$-Unification is decidable and of unification type infinitary.

Proof: Example 4.20 shows a pseudo-linear theory with a unification problem that leads to infinitely many cws. mgu's. The only crucial rule of sorted unification concerning termination and decidability is the rule Weakening. Assume that the rule is applied to an equation $x = t$ occurring in standard solved unification problem Γ. After the application of Weakening using declarations $S_i(t_i)$ standard unification is performed again. Let t'_i be the terms resulting from the application of Weakening and standard unification to $x = t$. As $\mathcal{L}$ is pseudo-linear we have $\max(Depth(t), Depth(t_i)) \geq Depth(t'_i)$ using Lemma 3.3 or a variable which occurs non-pseudo-linear in t is removed. If a variable occurring non-pseudo-linear is removed the number of non-pseudo-linear variable occurrences decreases for the t'_i. Whence the number of different terms modulo renaming that may be generated by the Weakening rule is finite, because there are only finitely many sort and function symbols in $\mathcal{L}$ and the maximal depth of generated terms is bound. Thus after a certain number of steps the unification problem is either solved or suspended. Therefore $\mathcal{L}$-Unification is decidable for pseudo-linear sort theories. □

The following example shows that the algorithm Sorted_Unification does not decide non-pseudo-linear sort theories. I conjecture sorted unification to be undecidable in general. This is not easy to prove because all techniques used so far rely on the empty sort problem. In this article the empty sort problem is separated from the unification problem.

Example 5.5 Consider the following sort theory and unification problem ($\mathcal{S}(x_i) = S$):

Sorted_Rule_Application(Γ_{WO},Γ_U)

1. *Result*: $= \emptyset$
2. **If** $\Gamma_U = \emptyset$ **Then** *Result*: $= \{(\Gamma_{WO}, \Gamma_U, \mathit{Solved})\}$, **Return**(*Result*)
3. Select an equation $x = t$ from Γ_U
4. **If** rule Sorted Fail is applicable to $x = t$ **Then** *Result*: $=$ $\{(\Gamma_{WO}, \Gamma_U, \mathit{Fail})\}$, **Return**(*Result*)
5. **If** $t \in \mathcal{T}^{\emptyset}_{\mathcal{S}(x)}$ **Then** *Result*: $= \{(\Gamma_{WO} \cup \{x = t\}, \Gamma_U \setminus \{x = t\}, \mathit{Unsolved})\}$, **Return**(*Result*)
6. **If** t is not a variable and $t \notin \mathcal{T}_{\mathcal{S}(x)}$ and $x = t$ occurs in Γ_{WO} modulo renaming **Then** *Result*: $= \{(\Gamma_{WO}, \Gamma_U, \mathit{Suspended})\}$, **Return**(*Result*)
7. **If** $t \in \mathcal{T}_{\mathcal{S}(x)}$ then if t is not a variable or t is a variable and $x \notin \mathcal{T}^{\emptyset}_{\mathcal{S}(t)}$ **Then** *Result*: $= \{(\Gamma_{WO} \cup \{x = t\}, \Gamma_U \setminus \{x = t\}, \mathit{Unsolved})\}$
8. **If** rule Subsort is applicable to $x = t$ **Then** *Result*: $=$ *Result* $\cup \{(\Gamma_{WO} \cup \{t = x\}, \Gamma_U \setminus \{x = t\}, \mathit{Unsolved})\}$
9. **If** rule Common Subsort is applicable to $x = t$ **Then** *Result*: $=$ *Result* $\cup$ $\{(\Gamma_{WO} \cup \{x = z, t = z\}, \Gamma_U \setminus \{x = t\}, \mathit{Unsolved})\}$ where $\mathcal{S}(z) = \mathcal{S}(x) \cup \mathcal{S}(t)$
10. **If** rule Weakening is applicable to $x = t$ where $t = f(t_1, \ldots, t_n)$ **Then** *Result*: $=$ *Result* $\cup \{(\Gamma_{WO} \cup \{x = t\}, \Gamma_U \setminus \{x = t\} \cup \{t_1 = s_{i,1}, \ldots, t_n = s_{i,n}\}, \mathit{Unsolved})\}$
11. **Return**(*Result*)

Table 4: Algorithm Sorted_Rule_Application

Sorted_Unification(PU,PP,*Resource*)

1. $PS := \emptyset$
2. **While** $PU \neq \emptyset$ and *Resource* > 0 **Do**
3. Select a pair (Γ_{WO}, Γ_U) from PU
4. $PU := PU \setminus \{(\Gamma_{WO}, \Gamma_U)\}$
5. (Γ_U, *Value*) := Standard_Unification(Γ_U)
6. **If** *Value* = *Solved* **Then**
7. *Resource* := *Resource* $- 1$
8. *Result* :=Sorted_Rule_Application(Γ_{WO}, Γ_U)
9. **For** each triple $(\Gamma'_{WO}, \Gamma'_U$, *Value*) in *Result* **Do**
10. **If** *Value* = *Solved* **Then** $PS := PS \cup \{\Gamma'_{WO}\}$
11. **If** *Value* = *Unsolved* **Then** $PU := PU \cup \{(\Gamma'_{WO}, \Gamma'_U)\}$
12. **If** *Value* = *Suspended* **Then** $PP := PP \cup \{(\Gamma'_{WO}, \Gamma'_U)\}$
13. **For** each problem Γ in PS **Do**
14. *Result* := Resume_Suspended_Problems(PP,Γ)
15. $PU := PU \cup$ *Result*
16. $PP := PP \setminus$ *Result*
17. **For** each unification problem Γ in PS **Do**
18. (Γ', *Value*) :=Standard_Unification(Γ)
19. **If** *Value* = *Solved* **Then** replace Γ in PS by Γ'
20. **Return**(PS, PU, PP)

Table 5: Algorithm Sorted_Unification

$$\begin{array}{lcl} \mathcal{L} & = & \{S(f(x_1, f(g(x_1), x_2))), S(g(x_1))\} \\ \Gamma & = & \{x_3 = f(g(x_4), x_5)\} \end{array}$$

Applying Sorted_Unification($\{(\emptyset, \{x_3 = f(g(x_4), x_5)\})\}$,$\emptyset$,$n$) results in the following values of the variables Γ_{WO} and Γ_U after the call of Standard_Unification at line 5.

$$\begin{array}{lrcl} 1. & \Gamma_{WO} & = & \emptyset \\ & \Gamma_U & = & \{x_3 = f(g(x_4), x_5)\} \\ 2. & \Gamma_{WO} & = & \{x_3 = f(g(x_4), x_5)\} \\ & \Gamma_U & = & \{x_6 = g(x_4), x_5 = f(g(g(x_4)), x_7)\} \\ 3. & \Gamma_{WO} & = & \{x_3 = f(g(x_4), x_5), x_6 = g(x_4)\} \\ & \Gamma_U & = & \{x_5 = f(g(g(x_4)), x_7)\} \\ 4. & \Gamma_{WO} & = & \{x_3 = f(g(x_4), x_5), x_6 = g(x_4), x_5 = f(g(g(x_4)), x_7)\} \\ & \Gamma_U & = & \{x_8 = g(g(x_4)), x_7 = f(g(g(g(x_4))), x_9)\} \\ 5. & \Gamma_{WO} & = & \{x_3 = f(g(x_4), x_5), x_6 = g(x_4), x_5 = f(g(g(x_4)), x_7), x_8 = g(g(x_4))\} \\ & \Gamma_U & = & \{x_7 = f(g(g(g(x_4))), x_9)\} \end{array}$$

There is no solution to the initial unification problem. The algorithm never terminates (except by resource bounding) because always new equations of the form $x_{2j+1} = f(g^j(x_4), x_{2j+3})$ were created by the rule Weakening.

6 Applications of Sorted Unification

Now we will fit the previous results together and apply it in full detail to the standard resolution calculus [20, 32] and the tableau calculus [24, 9]. As we promised in the introduction in order to make a calculus a sorted one, replace standard unification with sorted unification.

6.1 Resolution with Sorts

The starting point for the resolution calculus is a database of clauses Δ. From these clauses the sort theory $\mathcal{L}$ is chosen. $\mathcal{L}$:= $\{(S_i(t_i), C_i')\}$ such that for each declaration clause $C_i \in \Delta$ we choose exactly one declaration $S_i(t_i)$ with $C_i = \{S_i(t_i)\} \cup C_i'$. The database is modified by the resolution and factorization rule extended with sorted unification. If application of the rules derives new declaration clauses, $\mathcal{L}$ must be updated. On the other hand if a reduction rule (e.g. subsumption) removes a declaration clause, this clause can also be removed from $\mathcal{L}$. The inference rules are defined with respect to the dynamic sort theory $\mathcal{L}$.

Definition 6.1 (Inference Rules)
The rules are

$$\text{Resolution} \quad \frac{P(t_1, \ldots, t_n) \vee C_1 \qquad \neg P(s_1, \ldots, s_n) \vee C_2}{C_1\sigma \vee C_2\sigma \vee D \vee E}$$

where $\sigma^{\mathbf{c}} = (\sigma, D)$ is a cws. mgu of $P(t_1, \ldots, t_n)$ and $P(s_1, \ldots, s_n)$ and E is a set of literals (conditions) which guarantees the sorts occurring in $COD(\sigma)$ but not in $C_1\sigma \vee C_2\sigma \vee D$ to be non-empty. The non-emptiness conditions can be computed by Algorithm 4.22.

$$\text{Factorization} \qquad \frac{P(t_1, \ldots, t_n) \vee P(s_1, \ldots, s_n) \vee C}{P(t_1, \ldots, t_n)\sigma \vee C\sigma \vee D}$$

where $\sigma^{\mathbf{c}} = (\sigma, D)$ is a cws. mgu of $P(t_1, \ldots, t_n)$ and $P(s_1, \ldots, s_n)$.

The soundness of the rules follows immediately from their form, Lemma 4.4, and Lemma 4.18. Checking for empty sorts is not necessary for the factorization rule, because all sorts attached to variables in the codomain of σ occur in the factor.

Theorem 6.2 (Completeness Theorem for Resolution with Sorts) Let Δ be a clause database. We choose $\mathcal{L} := \{(S_i(t_i), C_i')\}$ such that for each declaration clause $C_i \in \Delta$ we choose exactly one declaration $S_i(t_i)$ with $C_i = \{S_i(t_i)\} \cup C_i'$.

If Δ is unsatisfiable there exists a derivation of the empty clause using resolution and factorization. The set $\mathcal{L}$ must be updated every time a new declaration clause is derived.

Proof: (Sketch) It can be proved that every standard refutation of Δ yields a cws. substitution with respect to all declarations occurring in Δ. Now selecting one declaration from each clause consisting of declarations only and updating $\mathcal{L}$ during the refutation corresponds to a case analysis. Putting this together with Lemma 4.19 we obtain completeness for resolution. For more details see Weidenbach [28]. □

Considering the example from the introduction

Δ:	
(1)	$S(a) \vee R(a)$
(2)	$S(f(x_1))$
(3)	$Q(y_1, y_1)$
(4)	$\neg Q(a, y_2)$
(5)	$P(x_2, f(a))$
(6)	$\neg P(a, x_3)$

where $\mathcal{S}(x_i) = S$ and $\mathcal{S}(y_i) = R$ we select the sort theory

$$\mathcal{L}_1 \quad = \quad \{(S(a), \{R(a)\}), S(f(x_1))\}$$

With respect to $\mathcal{L}_1$ the sort R is empty. The only applicable resolution step is (5)1 with (6)1 with cws. mgu

$$\sigma_1^{\mathbf{c}} \quad = \quad (\{x_2 \mapsto a, x_3 \mapsto f(a)\}, \{R(a)\})$$

resulting in the resolvent

(7) $R(a)$

A new declaration clause is derived. Therefore $\mathcal{L}_1$ must be updated. As (7) subsumes (1) the new sort theory is

$$\mathcal{L}_2 \quad = \quad \{R(a), S(f(x_1))\}$$

Now S is empty and the only possible resolution step is (3)1 with (4)1 with cws. mgu

$$\sigma_2^{\mathfrak{c}} \quad = \quad (\{y_1 \mapsto a, y_2 \mapsto a\}, \emptyset)$$

yielding the empty clause

(8) $\square$

The search space for Δ is finite. After selecting one of the two possible sort theories the application of resolution is deterministic. For Δ', the standard formalization of Δ this is not the case. Here the search space is infinite and resolution can be applied to several clauses ($\mathcal{S}(z_i) = \top$).

Δ':	
(1)	$S(a) \vee R(a)$
(2)	$\neg S(z_1) \vee S(f(z_1))$
(3)	$\neg R(z_2) \vee Q(z_2, z_2)$
(4)	$\neg R(z_3) \vee \neg Q(a, z_3)$
(5)	$\neg S(z_4) \vee P(z_4, f(a))$
(6)	$\neg S(z_5) \vee \neg P(a, z_5)$

6.2 Tableau with Sorts

The free variable tableau is modified to a free variable sorted tableau. The usual notions for tableau are used [24, 9]. Thus an α formula is one of the kind (possibly after moving a negation inwards) $\mathcal{F} \wedge \mathcal{G}$, a β formula has the form $\mathcal{F} \vee \mathcal{G}$, and the γ and δ formulae have a universal or an existential quantifier as top symbol respectively. Formulas of the form $\neg\neg\mathcal{F}$ can be seen as α formulas where $\alpha_1 = \alpha_2 = \mathcal{F}$.

Definition 6.3 (Inference Rules)
The inference rules for tableau consist of the expansion rules

$$\alpha \quad \frac{\alpha}{\begin{array}{c}\alpha_1\\ \alpha_2\end{array}} \qquad\qquad \beta \quad \frac{\beta}{\begin{array}{c|c}\beta_1 & \beta_2\end{array}}$$

$$\gamma \quad \frac{\gamma}{\gamma(x)} \qquad\qquad \delta \quad \frac{\delta}{\begin{array}{c}\delta(f(x_1, \ldots, x_n))\\ S_1(f(x_1, \ldots, x_n))\\ \vdots\\ S_m(f(x_1, \ldots, x_n))\end{array}}$$

where x is a new variable in the γ-rule. The function f is a new Skolem function where $x_1, \ldots, x_n$ are all the used free variables in the δ rule [9]. The sort $\{S_1, \ldots, S_m\}$ is the sort of the existentially quantified variable in δ which is replaced by $f(x_1, \ldots, x_n)$. Branches are closed atomically applying the atomic closure rule:

Atomic Closure	Let $\mathbf{T}$ be a tableau for a set of sentences Δ. If some branch in $\mathbf{T}$ contains A and $\neg B$, where A and B are atoms, then $\mathbf{T}\sigma$ is also a tableau for Δ, where σ is a most general non-empty well-sorted unifier of A and B.

The sort theory $\mathcal{L}$ selected for unification is the set of all declarations occurring on the current branch. Then σ must be a non-empty well-sorted unifier with respect to this sort theory. For these sort theories we always have $\mathcal{T}_S = \mathcal{T}_S^{\emptyset}$ for every sort S. Therefore the additional notions for conditional expressions can be skipped.

The δ rule not only generates the Skolemized formula but also the necessary declarations for the Skolem function. The semantics of existential quantifiers justifies the extended Skolemization (see the relativization rule in Section 2). We can easily prove that the Skolemized formula is equivalent to the original formula with respect to satisfiability.

Theorem 6.4 (Completeness Theorem for Tableau with Sorts) Let $\mathcal{F}$ be a valid sentence and $\mathcal{R}$ be any fair tableau construction rule (see Fitting [9]). Then there exists a closed tableau for $\neg\mathcal{F}$ such that:

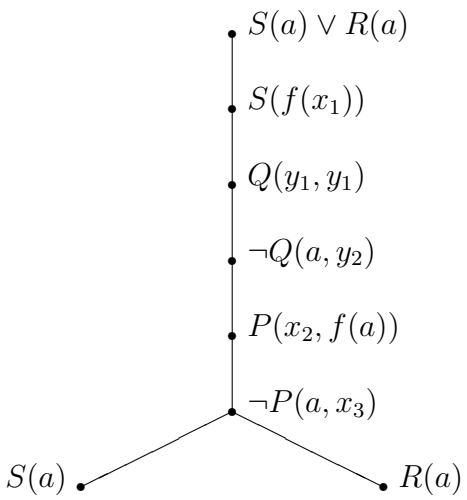
(i) All tableau expansion rules come first and are according to $\mathcal{R}$.

(ii) A single tableau substitution rule application follows, using a substitution σ that is a most general non-empty well-sorted atomic closure substitution.

Proof: The proof is a straight forward extension of the proof given by Fitting [9]. As we have Lemma 4.5 for sort theories used in tableau ($\mathcal{T} = \mathcal{T}^{\emptyset}$), the non-empty well-sorted unifiers have the same properties with respect to semantics than standard unifiers. Thus together with Lemma 4.19 we obtain completeness of tableau with sorts. □

In fact, Theorem 6.4 is the sorted generalization of Theorem 7.8.6 given by Fitting [9, p. 179]. The application of sorted unification to tableau is that simple, because in tableau a case analysis is explicitly done by the β-rule. Therefore no conditional declarations are needed in the sort theory, whence Lemma 4.5 holds. Note that in different branches of a tableau different sort theories are considered in general.

Eventually, we solve the example from the introduction with the new tableau method. The tableau $\mathbf{T}$ after application of the expansion rules is



where $\mathcal{S}(x_i) = S$ and $\mathcal{S}(y_i) = R$. Applying atomic closure to the first branch, the corresponding sort theory is

$$\mathcal{L}_1 \quad = \quad \{S(a), S(f(x_1))\}$$

There are only two literals which become complementary: $P(x_2, f(a))$ and $\neg P(a, x_3)$ with cws. unifier

$$\sigma_1 \quad = \quad \{x_2 \mapsto a, x_3 \mapsto f(a)\}$$

The second branch can be closed applying atomic closure to $Q(y_1, y_1)$ and $\neg Q(a, y_2)$ with respect to the sort theory

$$\mathcal{L}_2 \quad = \quad \{R(a), S(f(x_1))\}$$

and cws. unifier

$$\sigma_2 \quad = \quad \{y_1 \mapsto a, y_2 \mapsto a\}$$

As σ_1 and σ_2 are variable disjoint, $\mathbf{T}\sigma_1\sigma_2$ is the closed tableau for the input formula and $\sigma_1\sigma_2$ the necessary tableau substitution (see Theorem 6.4).

7 Discussion

The approach of Schmidt-Schauß [22] extends Walther's work [27]. This paper generalizes the work of Schmidt-Schauß along the following dimensions:

(i) A sort is not only a sort symbol but a set of sort symbols.

(ii) Sorts may denote the empty set.

(iii) Declarations occurring in the sort theory may be conditioned by other literals.

(iv) If applied to a calculus the sort theory is a dynamic part of the database.

(i) is an extension of the sort language. It allows for a finite set of mgu's in elementary sort theories where Schmidt-Schauß unification algorithm computes infinitely many mgu's (Lemma 4.21). Schmidt-Schauß considered all (base) sorts to be a priori non-empty (ii). For the sorted unification theory this restriction plays no role. The problem arises if sorted unification is applied to a calculus. The precise problem is Skolemization. Traditionally, Skolemization in sorted languages is done in the following way [22, 11]. The formula $\exists x_S \mathcal{F}$ contained in some formula $\mathcal{G}$ is replaced by $\mathcal{F}\{x_S \mapsto f(\ldots)\}$ where f is the new Skolem function in the respective variables and the declaration $S(f(\ldots))$ is put outside $\mathcal{G}$ into the sort theory. This is only a sound operation if S is not empty. Otherwise local Skolemization [28] must be performed where $\exists x_S \mathcal{F}$ is replaced by $\mathcal{F}\{x_S \mapsto f(\ldots)\} \wedge S(f(\ldots))$. Now declarations may occur together with other literals. Therefore the sort theory must either be a dynamic part of the database or additional inference rules which form the bridge between sort literals in the database and sort literals in the sort theory become necessary. Thus allowing for empty sorts results either in a calculus consisting of more rules than the standard rules of the calculus (e.g. see Cohn [5] or Beierle et al. [1]) or the sort theory must be a dynamic part of the database (iv) (see Weidenbach [30]). Thus approaches with a static sort theory and no additional inference rules must require sorts to be a priori non-empty (e.g. see Schmidt-Schauß [22] or Frisch [11]).

The same that holds for the introduction of possibly empty sorts applies to the notion of conditional expressions (iii). The additional notion does not change general properties of sorted unification (see Lemma 4.10). But then applied to the resolution calculus the notion allows an extension without the introduction of new inference rules. Until now this is the most efficient (in terms of search space reduction) method known [30].

Uribe [26] showed that sorted unification is decidable for semi-linear sort theories. Lemma 5.4 extends the class of sort theories with a decidable unification problem to pseudo-linear sort theories. However, there are differences between the two approaches. I separate the problem of sorted unification from the problem whether a sort is empty. Uribe doesn't separate these problems. Whence he both proved unification and the empty sort problem decidable for semi-linear sort theories. Lemma 5.4 only says that the unification part is decidable for pseudo-linear

sort theories. In fact, the empty sort problem is not decidable for pseudo-linear sort theories. This was shown by Tommasi [25]. Deciding whether a sort is empty (or whether two base sorts share a common ground term) is a harder problem than unification in sort theories. Uribe also gives a finite representation for a solution of a unification problem in semi-linear sort theories. The sorted unification problem for semi-linear sort theories is of unification type infinitary (see Example 4.20).

For his result, Tommasi used the correspondence between sort theories and finite tree automaton. This correspondence was first pointed out by Comon [6]. It makes results in automaton theory available to unification in sort theories. The general undecidability result given by Schmidt-Schauß could be confirmed. In addition, it was possible to proof decidability for linear sort theories this way. Until now other researchers have used the correspondence between tree automata and theories related to sort theories (e.g. set constraints) [3, 13].

The paper given by Frisch and Cohn [12] reformulates and abstracts results previously established by Schmidt-Schauß. They only consider base sorts as sorts but abstract away from a specific concept of a sort theory. An oracle is assumed which can be asked whether terms are well-sorted and which computes all possible weakenings σ such that $t\sigma \in \mathcal{T}_S$ for some sort S. Thus their sorted unification procedure GSUP has no specific sorted rules but a general, abstract weakening rule. Weakening and all cases of sorted failure are left to this rule. Compared to Schmidt-Schauß unification algorithm GSOUP, the six sorted rules are comprised in this abstract weakening rule. All other rules of the two algorithms are identical.

A property which was very often addressed in the past is regularity of sort theories [22, 14]. Schmidt-Schauß showed that if a sort theory is regular a more efficient unification algorithm can be formulated. However, regularity is an undecidable property for a given sort theory. The sorted unification algorithm presented in this paper overcomes the problem. It is as efficient as the unification algorithm SOUP for regular sort theories given by Schmidt-Schauß. This was possible by introducing sets of sort symbols as sorts. Thus from an efficiency point of view regularity is a superfluous concept. In addition, Comon [7] pointed out that if equality is included in the logic regularity is also not a useful concept.

In my previous work [31, 28, 30] I mainly addressed the extension of resolution with sorted unification. Only base sorts were considered as sorts and no specific results were given for unification in sort theories. In this paper the notion of sorts is extended, unification in elementary sort theories is proved to be decidable and of unification type finitary. Unification in pseudo-linear sort theories is proved decidable and of unification type infinitary. Algorithms for the computation of empty sorts and well-sortedness are presented and upper bounds for their time complexity established. A transformation of the rule based sorted unification algorithm into a sorted unification procedure is given. The new representation of the algorithm introduces more control structure and more data structure. The additional structure is used to prove unification in pseudo-linear sort theories decidable. Eventually, I generalized tableau with sorts from tableau with free variables. Future work will

concentrate on the combination of sorts and equality.

References

[1] C. Beierle, U. Hedstück, U. Pletat, and J. Siekmann. An order-sorted logic for knowledge representation systems. *Artificial Intelligence*, 55:149–191, 1992.

[2] M. Bidoit and J. Corbin. A rehabilitation of robinson's unification algorithm. In *Proceedings of IFIP 9th World Computer Congress*, pages 909–914. North-Holland, 1983.

[3] B. Bogaert and S. Tison. Equality and disequality constraints on direct subterms in tree automata. In *Proc. of 9th Annual Symposium on Theoretical Aspects of Computer Science, STACS92*, pages 161–171. Springer Verlag, 1992.

[4] A.G. Cohn. A more expressive formulation of many sorted logic. *Journal of Automated Reasoning*, 3(2):113–200, 1987.

[5] A.G. Cohn. A many sorted logic with possibly empty sorts. In *11th International Conference on Automated Deduction, CADE-11, LNCS 607*, pages 633–647. Springer Verlag, 1992.

[6] H. Comon. Inductive proofs by specifications transformation. In *Proc. of RTA*, pages 76–91. Springer Verlag, 1989.

[7] H. Comon. Equational formulas in order-sorted algebras. In *Proc. of ICALP*, pages 674–688. Springer Verlag, 1990.

[8] N. Dershowitz. Termination of rewriting. *Journal of Symbolic Computation*, 3(1):69–115, July 1987.

[9] M. Fitting. *First-Order Logic*. Texts and Monographs in Computer Science. Springer, 1990.

[10] A.M. Frisch. A general framework for sorted deduction: Fundamental results on hybrid reasoning. In *Proceedings of the First International Conference on Principles of Knowledge Representation and Reasoning*, pages 126–136, May 1989.

[11] A.M. Frisch. The substitutional framework for sorted deduction: fundamental results on hybrid reasoning. *Artificial Intelligence*, 49:161–198, 1991.

[12] A.M. Frisch and A.G. Cohn. An abstract view of sorted unification. In *11th International Conference on Automated Deduction, CADE-11, LNCS 607*, pages 178–192. Springer Verlag, 1992.

[13] R. Gilleron, S. Tison, and M. Tommasi. Solving systems of set constraints using tree automata. In P. Enjalbert, K.W. Wagner, and A. Finkel, editors, *Proc. of 10th Annual Symposium on Theoretical Aspects of Computer Science, STACS93*, pages 505–514. Springer Verlag, February 1993.

[14] J.A. Goguen and J. Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. Draft, Programming Research Group, University of Oxford and SRI International, 1989.

[15] J. Herbrand. Investigations in proof theory: The properties of true propositions. In J. van Heijenoort, editor, *From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931*, pages 525–581. Harvard University Press, 1967.

[16] G. Huet. Confluent reductions: Abstract properties and applications to term rewriting systems. *Journal of the ACM*, 27(4):797–821, 1980.

[17] A. Martelli and U. Montanari. An efficient unification algorithm. *ACM Trans. Programming Languages ans Systems*, 4(2):258–282, 1982.

[18] A. Oberschelp. Untersuchungen zur mehrsortigen quantorenlogik. *Mathematische Annalen*, 145:297–333, 1962.

[19] M. Paterson and M. Wegman. Linear unification. *Journal of the Computers and Systems*, 16, 1978.

[20] J.A. Robinson. A machine-oriented logic based on the resolution principle. *Journal of the ACM*, 12(1):23–41, January 1965.

[21] A. Schmidt. Über deduktive theorien mit mehreren sorten von grunddingen. *Mathematische Annalen*, 115:485–506, 1938.

[22] M. Schmidt-Schauß. *Computational aspects of an order sorted logic with term declarations*, volume 395 of *Lecture Notes in Artificial Intelligence*. Springer Verlag, 1989.

[23] J. Siekmann. Unification theory. *Journal of Symbolic Computation, Special Issue on Unification*, 7:207–274, 1989.

[24] R.M. Smullyan. *First-Order Logic*. Springer Verlag, 1968.

[25] M. Tommasi. Automates avec tests d'égalités entre cousins germains. Master's thesis, Universié de Lille, France, 1991.

[26] T.E. Uribe. Sorted unification using set constraints. In *11th International Conference on Automated Deduction, CADE-11, LNCS 607*, pages 163–177. Springer Verlag, 1992.

[27] C. Walther. *A Many-sorted Calculus based on Resolution and Paramodulation.* Research Notes in Artificial Intelligence. Pitman Ltd., 1987.

[28] C. Weidenbach. A sorted logic using dynamic sorts. MPI-Report MPI-I-91-218, Max-Planck-Institut für Informatik, Saarbrücken, December 1991.

[29] C. Weidenbach. A new sorted logic. In *Proceedings of the 16th German AI-Conference, GWAI-92*, pages 43–54. Springer, LNAI 671, April 1992.

[30] C. Weidenbach. Extending the resolution method with sorts. In *Proc. of 13th International Joint Conference on Artificial Intelligence, IJCAI-93.* Morgan Kaufmann, 1993. To appear.

[31] C. Weidenbach and H.J. Ohlbach. A resolution calculus with dynamic sort structures and partial functions. In *Proceedings of the 9th European Conference on Artificial Intelligence*, pages 688–693. Pitman Publishing, London, August 1990.

[32] L. Wos, R. Overbeek, E. Lusk, and J. Boyle. *Automated Reasoning, Introduction and Applications.* McGraw-Hill, second edition, 1992.